Pyragrid: Bringing Peer-to-Peer and Grid Architectures Together

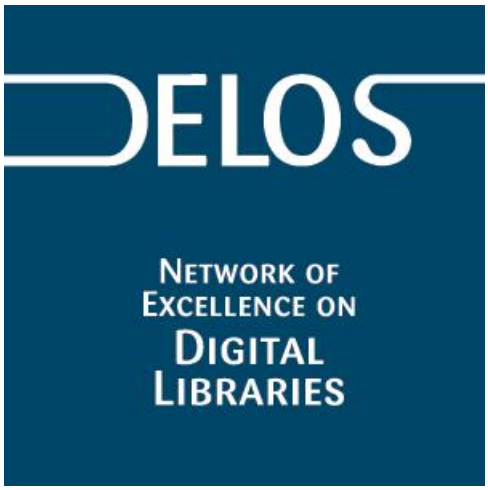
Maristella Agosti
Hans-Jörg Schek
Can Türker (Eds)

Digital Library Architectures:

Peer-to-Peer, Grid, and Service-Orientation

Pre-proceedings of the Sixth Thematic Workshop of the EU Network of Excellence DELOS

S. Margherita di Pula, Cagliari, Italy
24-25 June, 2004



Edizioni Libreria Progetto Padova

Preface

DELOS is an interdisciplinary EU FP6 Network of Excellence with a broad vision: Future digital libraries should enable any citizen to access human knowledge any time and anywhere, in a friendly, multi-modal, efficient and effective way. The main objective of DELOS is to define and conduct a joint program of activities in order to integrate and coordinate the ongoing research activities of the research teams in the field of digital libraries for the purpose of developing next generation digital library technologies.

Various newer computing areas like peer-to-peer, grid, and service-oriented computing provide new opportunities and challenges for architectures of future digital libraries. Peer-to-peer data management allows for loosely coupled integration of information services and sharing of information such as recommendations and annotations. Grid computing middleware is needed because certain services within digital libraries are complex and computationally intensive, e.g., extraction of features in multimedia documents to support content-based similarity search or for information mining in bio-medical data. The service-orientation provides mechanisms to describe the semantics and usage of information services and to combine services into workflow processes for sophisticated search and maintenance of dependencies. It is obvious that elements of all three directions should be combined in a synthesis for future digital libraries architectures.

This volume contains the papers accepted for the Sixth Thematic Workshop of the EU Network of Excellence DELOS on Digital Library Architectures, S. Margherita di Pula (Cagliari), Italy, 24-25 June, 2004. This workshop is co-located with the 12th Italian Symposium on Advanced Database Systems (SEBD 2004) and it is organized jointly by the DELOS Network of Excellence and the Department of Information Engineering of the University of Padua. The accepted workshop papers address a broad range of issues in digital library architectures, and thus will provide many starting points for interesting discussions.

Finally, we would like to thank all the persons that have contributed directly or indirectly to the organization of this workshop. Specifically, we want acknowledge much help by Nicola Ferro, Emma Liere, Francesca Borri, and Bruno Le Dantec. Last but not least, we thank the members of the program committee for helping in selecting the papers as well as the authors who have submitted a paper to this workshop.

Maristella Agosti, Hans-Jörg Schek, and Can Türker June 2004

Program Committee

Maristella Agosti	University of Padua, Italy
Elisa Bertino	University of Milano, Italy
Donatelli Castelli	CNR-ISTI, Italy
Stavros Christodoulakis	Technical University of Chania, Greece
Wilhelm Hasselbring	OFFIS Oldenburg, Germany
Yannis Ioannidis	University of Athens, Greece
Martin Kersten	CWI Amsterdam, Netherlands
Liz Lyon	UKOLN Bath, United Kingdom
Erich Neuhold	FhG Darmstadt, Germany
Hans-Jörg Schek	ETH Zurich, Switzerland/UMIT Innsbruck, Austria; chair
Heiko Schuldt	UMIT Innsbruck, Austria
Can Türker	ETH Zurich, Switzerland; co-chair
Gerhard Weikum	MPI Saarbrücken, Germany
Pavel Zezula	Masaryk University Brno, Czech Republic

Links

Web page of the workshop:	http://www.dbs.ethz.ch/delos/
Web page of the DELOS project:	http://www.delos.info/

Table of Contents

Advanced Services for Digital Libraries

Service Oriented Architecture

Workflow and Query Tasks

Pyragrid: Bringing Peer-to-Peer and Grid Architectures Together

Stratis D. Viglas

School of Informatics
University of Edinburgh, UK
sviglas@inf.ed.ac.uk

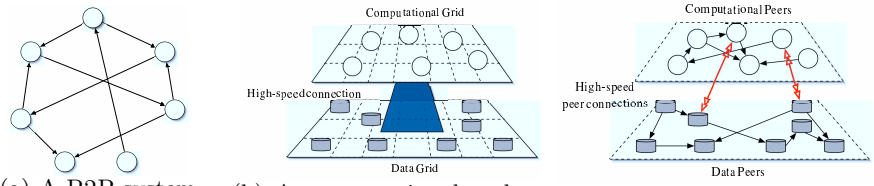
Abstract. Peer-to-peer and Grid architectures share common abstractions with respect to storage and computing functionality. To date, however, there has been little insight on how the architectures can be bridged. The purpose of this paper is to start the discussion on how this can be achieved. In particular, we present a system called Pyragrid and its architecture. The system borrows the principle of locating data used in a peer-to-peer system and extends it to more complex functionality and types of processing. The Pyragrid architecture aims at capturing generic distributed computing paradigms, irrespective of their specific field. As an instance of a system built on top of the Pyragrid architecture, we present PyraDB, a novel distributed database system.

1 Introduction

In the past few years there has been an evident research turn towards distributed computing. Peer-to-peer systems (P2P) are the token distributed system architecture, focussing mainly on exploiting the huge storage infrastructure provided by a collection of interconnected machines. The P2P premise is *collaborative sharing*: everything is available to everyone, so long as there are ways of accessing the information. On the other hand, Grid computing has emerged as the most serious contender for scientific and computation-intensive applications. The Grid premise is *collaborative computation*: every machine in the Grid can undertake computation for everyone, so long as there are computational cycles to spare. Schematic descriptions of the two architectures are presented in Fig. 1(a) and Fig. 1(b). The purpose of this paper is to start the discussion on bridging the two areas. Though the two areas seem diverse at first inspection, they have rather complementary functionalities, and this is what we will attempt to exploit.

A P2P system is rather chaotic in nature[1] as there is no clear and static concept of a network. Machines in a P2P system, referred to as *peers*, join and leave the network at will. Each time they join new data becomes available; each

[1] There is, of course, the concept of structured P2P systems, but a pure P2P system is one that is not organised; rather, peers join and leave on demand and a static structure is never assumed. In subsequent discussion we will only refer to pure P2P systems.



(a) A P2P system (b) A computational and a data grid (c) The Pyragrid architecture

Fig. 1. P2P, Grid, and PG architectures

time they leave data ceases existing. The situation, however, is not always as clear since, when a peer leaves the network, there is a high probability that its data has been "shared" with some other peer; so, the data continues being "live" even after the peer that introduced it into the network leaves. The challenges in this scenario are making sure data is available, and minimizing the time needed to access the data by providing fast lookup functionality. At the same time the system needs to be load-balanced, so that peers are not overloaded with retrieval requests, and fault-tolerant, in the sense that peers joining and leaving the network should not affect its performance.

A Grid architecture can be conceptually decomposed into two layers: the *data grid* and the *computational grid*. The data grid presents a massive, interconnected storage infrastructure, connected to the computational grid via a high-speed network connection. The computational grid is the virtual single machine created by bringing together a large collection of independent hosts. Various ways of collaboration can be implemented on top of the virtual machine. The interconnection network at the computational grid level allows for communication across hosts, while the high-speed data connection to the data grid layer allows for data availability and fast exchange of information.

The greatest challenge in bridging the two architectures is combining the chaotic nature of a P2P system with the structured approach of the Grid. To do so, one needs to look at the big picture and identify not the architectural differences, but rather the shared objectives of the two frameworks. The conclusion is that a P2P system can act as the data grid of a larger Grid architecture. At the same time, the collaborative nature of the Grid can enrich the functionality of a P2P system. What we propose is a hybrid architecture, which we term *Pyragrid*[2] (PG), shown in Fig. 1(c). The PG architecture aims at using the P2P functionality as the infrastructure for both the computational and the data layers of the Grid. Collaboration in PG is achieved through the use of a high-level processing abstraction, termed the *Operator Graph Model* (OGM). It is used to capture the operations to be undertaken by a complex processing request and decompose it into problems of lower complexity. These sub-problems will be solved in

[2] The name is a euphemism for "Peer" and "Grid" combined in a single well-sounding word.

a collaborative fashion. The lookup functionality of P2P systems will be used to identify relevant data and computing resources.

As a first instance of the PG architecure, we envision a distributed database system prototype: PyraDB. The physical algebra employed by contemporary database systems will be mapped to the OGM, allowing for queries issued to the system to be decomposed in terms of the OGM. Relevant lookup operations to locate the necessary sources will be performed and the query will be evaluated in a collaborative fashion. By employing P2P primitives, the system affords to be completely dynamic in terms of data and computing resources being introduced, while at the same time guaranteeing load balancing and fault tolerance.

The remaining of this paper is organized as follows. Introductory background information on distributed processing using P2P or Grid systems is presented in Section 2. The PG architecture is presented in more detail in Section 3, while the mapping to PyraDB is presented in Section 4. Finally, related work is presented in Section 5, while conclusions are drawn in Section 6.

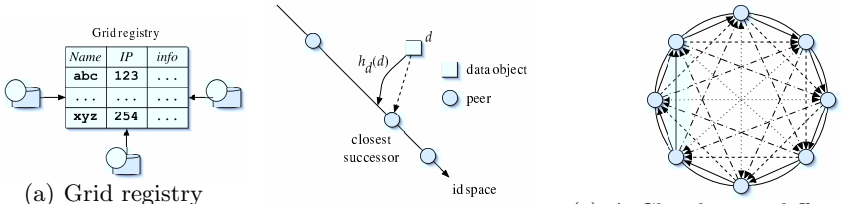
2 Background Information

In this section we will provide some more detail into the underlying structures of both Grid and P2P systems. In particular, we will concentrate on the lookup functionality of both.

2.1 Grid Information Service

A basic assumption of Grid architectures is that the system is highly structured. Though there are many proposals on Grid information systems [2], the idea is always the same. Computers participating in a Grid employ the information of a registry so that the remaining Grid participants can be located. This is not much different than what happens in traditional data networks.

The idea translates to the Grid, since a Grid can be thought of a specialized data network. Since it is specialized, however, there is extra information that needs to be monitored, such as processing capabilities of the participating machines; various annotations pertinent to the workflows that are taking place; and information on process migration just to name a few. An instance of such a service is shown in Fig. 2(a) where we have chosen to present the information as a relational table containing entries on the symbolic name of machines, their IP addresses in the Grid and various types of information. This is merely one way of implementing Grid information services; many others exist. To date there has not been a single standard as to how Grid information services are to be realized. All proposals, however, share the same concept of a registry that contains all relevant information about the machines of the Grid. The registry need not be centralized, it could be distributed *a la* DNS; the concept, however, is that, virtually, a well-defined central information service exists.



(a) Grid registry

(b) Mapping of data objects to peers in a DHT

(c) A Chord ring; different line styles denote different connection classes

Fig. 2. Grid information services, DHT principles and the Chord overlay

2.2 Distributed Hash Tables

One key idea behind a P2P system is a *distributed catalog*, *i.e.,* a way of quickly locating data in the network. The best-known example of such a structure is a *distributed hash table* (DHT). A DHT essentially maps a data object to the peer of the system that is responsible for it. The process, presented in Fig. 2(b), is the following:

- Peers are mapped to a discrete domain, which contains much more elements than the number of peers in the system; this domain is referred to as the *id space* $\mathcal{I}$ of the DHT. Mapping is done in terms of a hash function h_p. Assuming that there is a domain $\mathcal{P}$ of peers in the system, then the hash function h_p is defined as $h_p : \mathcal{P} \rightarrow \mathcal{I}$.
- The data objects are hashed by employing a different hash function h_d that maps them to the same id space $\mathcal{I}$. Assuming a data domain $\mathcal{D}$, the hash function h_d can be defined as $h_d : \mathcal{D} \rightarrow \mathcal{I}$.
- For a data object $d \in \mathcal{D}$, the peer $p \in \mathcal{P}$ is chosen to store it, such that $h_p(p) - h_d(d)$ is minimized. In other words, p is the closest successor of d in the id space.

The DHT is the principle behind mapping data objects to peers in a P2P system. What remains is an efficient way of routing requests to peers in the system. This is the *overlay network* structure of the system, *i.e.,* a virtual network of connections imposed over the peers of the system. Perhaps the best-known example of such an overlay network is the *Chord* ring [13], shown in Fig. 2(c). The concept behind a Chord ring is that each peer in the overlay network is connected to $\log m$ other peers, where m is the total number of peers in the system. There are $\log m$ connection classes altogether. Each of the $\log m$ connections of each peer belongs to a class, according to what the distance between the originating peer and the destination peer is. For example, in Fig. 2(c), there are four connection classes (shown with different types of lines): one for connections between peers with a distance of one, one for connections between peers with a distance of two, and so on. Routing is performed in terms of these connections

following a simple rule: at most one class of connection can be used to route a message from originating peer p_o to destination peer p_d. This means that at most $\log m$ "hops" will be made to route a single message. Chord rings also provide other guarantees with respect to fault tolerance and load balancing.

The PG architecture, as it will be presented in subsequent sections, is largely independent of the overlay network. It only assumes the existence of a DHT; it can therefore be used by any overlay network structure, as long as that structure provides DHT-style lookup functionality.

3 Pyragrid Architecture

In this section we will concentrate on the envisioned architecture for Pyragrid. The premise of any distributed processing paradigm has been that it is easier to move the processing to the data than vice versa. It is therefore the same premise that fuels the PG architecture. Of course, it is inevitable that some data will have to be moved around; the objective, however, it to minimize data movement.

The key idea is using a single namespace for three concepts that have been traditionally treated as distinct: peers, data and operations over the data. This is achieved is by use of what is termed the PG *Operator Graph Model* (OGM). As we will see, this leads to a uniform way of characterizing both the location of the data in the system's namespace, as well as the local processing that needs to be performed on the data. Localizing processing is achieved through decomposing larger operations into basic operations that can be locally evaluated.

3.1 High-level Description

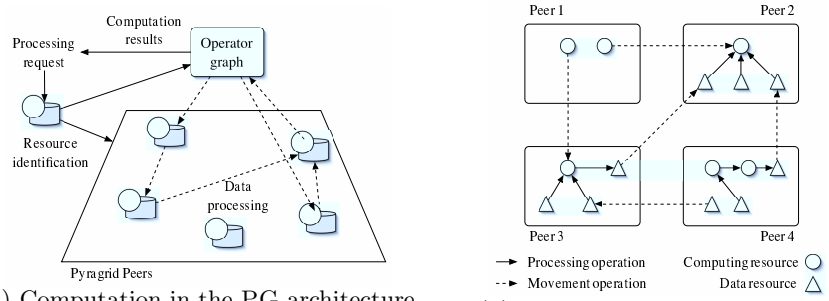
In this section we will present a high-level description of the PG architecture. In the next section we will move on to the details of the PG OGM that enables the realization of the architecture.

Central to the PG architecture is the concept of a *resource*. We assume there are two basic types of resources available:

- *Data resources*: The data stored by each peer. The storage infrastructure is a DHT, which allows for each peer to route requests for a particular data object to the peer that is responsible for storing it.
- *Computing resources*: The processing capabilities of each peer. Computing resources can be thought of as functional ways of manipulating the data resources. There is a fixed set of primitive computing resources that can be composed to allow for more complex processing.

The separation of resources into data-centric and computation-centric allows the system to have a clear and consistent way of addressing each request that it receives. Given this modeling, the system undertakes three main tasks:

1. Given data retrieval requests, identify the peers that are responsible for the data that is to be retrieved.



(a) Computation in the PG architecture

(b) The operator graph for a complex computation

Fig. 3. Schematics of how processing requests are handled in PG

2. Given computational requests, identify the peers that are capable of performing them.
3. Combining the above two in order to process complex computational problems. This means that either computing resources are shipped to peers that contain the relevant data, or vice versa. Of course, the first alternative is preferable.

The underlying principle of tackling the enumerated tasks is a generalized version of a DHT, as this was explained in Section 2.2. In more detail, all resources, independent of their type, are hashed and mapped to the peers of the system. Data resources are hashed on the types of their values, as well as the values themselves. Each computing resource has a name, and an abstract representation of the types of inputs it accepts and outputs it produces. This is called the *signature* of the computing resource. The id space in which peers, data resources and computing resources are mapped is common for all three types of entities, so the DHT principle is satisfied.

Each complex request is decomposed in terms of the OGM (further explained in Section 3.2) and routed to the peers of the system for processing. There is a fixed set of primary computing resources that can be made available by the peers. As in the case of data resources, not all peers are expected to have all computing resources available. The entire process is presented in Fig. 3(a). Each processing request is handled in two steps. First, an analysis of the request is performed at the peer receiving it, which results in a mapping of the request to the OGM. This triggers a number of DHT lookup operations so that the peers containing the relevant data and computing resources are identified. This leads to the building of the *operator graph* for the processing request, which is a roadmap of how the processing request can be satisfied by the system in terms of data and computing resource movement, as well as processing. The second step involves the actual computation, where the operator graph is traversed and the results of the computation are shipped back to the requesting entity.

A schematic of an operator graph is presented in Fig. 3(b), while it will be more formally defined in Section 3.2. Circles indicate computing resources, while triangles indicate data resources. Solid arrows indicate processing steps, while dashed arrows indicate resource movement. Finally, boxes indicate local processing. The first step in traversing the operator graph is shipping available resources to nodes so that computation can be initiated. Local processing then takes place in parallel across the peers, with necessary intermediate result movements acting as "joining" points for the computation.

In this representation, there is a "hidden" optimization problem. The graph is set up in such a way so that the total resource movement (whether it is data or computing resource movement, or even movement of intermediate computational results) is minimized. This implies a cost-based optimization modeling. The system uses an objective function that takes into account the relative sizes of data and computing resources, as well as the (possibly estimated) sizes of intermediate computational results.

3.2 Pyragrid Operator Graph Model

The PG processing model is based on the PG OGM. The OGM is an abstraction of all the operations that are necessary in order to satisfy a complex processing request. We assume there are three domains in the system: (i) the peer domain $\mathcal{P}$, which is the domain peer identifiers are derived from, (ii) the data domain $\mathcal{D}$, which is the domain data resource identifiers are derived from, and (iii) the computing domain $\mathcal{C}$, which is the domain computing resource identifiers are derived from.

There are two types of operators in the OGM: (i) *movement operations*, that model the movement of data or computing resources between peers; and (ii) *processing operations* that capture all the processing steps needed to address the request. The OGM is a graph comprised of various operations of each type. In the next paragraphs we will formally define each.

Movement Operators The movement operators capture the parts of the OGM that denote data or computing resource movement. The generic class of movement operators is denoted as $\mathcal{M}$ and there are two subclasses. Subclass $\mathcal{M}_D$, henceforth referred to as the data movement subclass, denotes data resource movement, while $\mathcal{M}_C$, henceforth referred to as the computing movement subclass denotes computing resource movement. To present a compact representation of the operators, we will use the following notation, which captures preconditions and postconditions:

$$(pre) \quad =: \quad op \quad := \quad (post)$$

The semantics are that before operator *op* has been applied preconditions *pre* hold. After operator *op* is applied postconditions *post* hold as well. The preconditions continue to hold, unless stated otherwise. Additionally, the notation $r \lhd p$ denotes that resource r is available at peer p.

Operators of the data movement subclass all have the same signature. Each operator $\mu_D \in \mathcal{M}_D$ of the subclass accepts as input a triplet consisting of a data resource identifier $d \in \mathcal{D}$ and two peer identifiers. The first peer identifier $p_o \in \mathcal{P}$ is the originating peer, while the second peer identifier $p_d \in \mathcal{P}$ is the destination peer. The data resource is available at the originating peer. After the operation the data resource is available at the destination peer. Using our notation, this can be expressed as:

$$d \in \mathcal{D}, p_o \in \mathcal{P}, p_d \in \mathcal{P}, d \lhd p_o \quad =: \quad \mu_D(d, p_o, p_d) \quad := \quad d \lhd p_d$$

The corresponding can be written for each operator $\mu_C \in \mathcal{M}_C$ of the computing movement subclass:

$$c \in \mathcal{C}, p_o \in \mathcal{P}, p_d \in \mathcal{P}, c \lhd p_o \quad =: \quad \mu_C(c, p_o, p_d) \quad := \quad c \lhd p_d$$

Processing Operators Processing operators can be thought of as manipulators of the data resources of a peer. We assume there is a class $\mathcal{R}$ of available processing operators $\mu_R \in \mathcal{R}$. Each operator is executed in one of the peers of the system and accepts as input the peer identifier and a set D_i of local data resources that it manipulates. It produces a new set of data resources D_o as output. The resulting data resources are local to the peer. Therefore, the signature of all processing operators can be expressed as:

$$D_i \subset \mathcal{D}, p \in \mathcal{P}, \forall d \in D_i(d \lhd p) \quad =: \quad D_o = \mu_R(D_,p) \quad := \quad \forall d \in D_o(d \lhd p)$$

Operator Graph Composition The final step is composing the operator graph for a processing request that the system accepts. The modeling we use is the following. A processing request is accepted at one of the peers. The request is analyzed so that the set of necessary data resources $d_i \in \mathcal{D}$ and the set of necessary computing resources $c_i \in \mathcal{C}$ are identified. Each data resource is denoted by a triangle in Fig. 3(b)'s graph, while each computing resource is denoted by a circle. The identification of resources leads into the forming of the "movement" part of the graph, which is denoted by the dashed arrows in Fig. 3(b). Each dashed arrow corresponds to a data movement operator, either μ_D or μ_C depending on the subclass of movement operation the operator belongs to (correspondingly, $\mathcal{M}_D$ or $\mathcal{M}_C$).

The next step is combining the computing resources in order to form processing operations over the data of the system. This is depicted by solid arrows in Fig. 3(b). Note that the result of a processing operation may be directly "fed" into a subsequent processing operator (*e.g.,* Peer 4 in Fig. 3(b)'s operator graph), or stored as a new data resource. The data resource may then be further moved to another peer so that the complete operator graph is formed (*e.g.,* the movement from Peer 3 to Peer 4 in Fig. 3(b)'s operator graph).

4 PyraDB: A Pyragrid Database Prototype

The first step towards addressing the viability of the PG approach is to map it to a well-defined processing paradigm. The paradigm so chosen is a database system. The reasons for this choice are the following:

- A database system is traditionally expected to store massive amounts of data; the same is expected of a data grid or a P2P system. It seems natural to build a database system on top of an architecture like PG, which combines the two.
- There is a clear mapping from the PG OGM to the physical algebra used by database systems. The algebraic operators, which form the procedural way of expressing relational queries, become PG operators.
- Old ideas of relational algebraic operator decomposition [3, 4], coupled with recent advances in data stream query evaluation [9, 15], allow for completely asynchronous and assymetric evaluation of queries. The consequence is that it is easier to decompose queries into in a way that facilitates local computation.

In the next few sections we will present how the PG OGM can be mapped to query evaluation algorithms and to the execution model employed by database systems in general. We will also discuss some initial thoughts on query optimization in PyraDB, as well as system evolution.

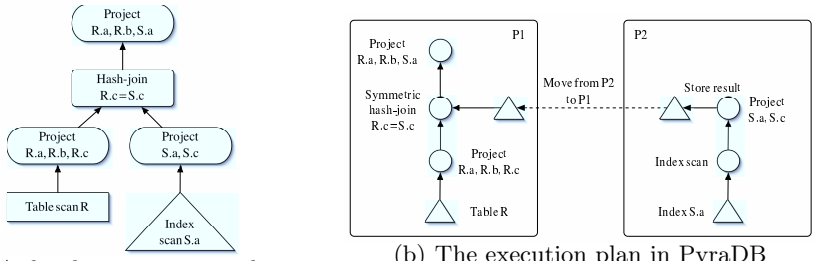
4.1 Query Execution

Query execution has been traditionally addressed in terms of an execution plan, *i.e.,* a tree that captures the physical operators used in the evaluation of a query, along with their interconnections. An instance of an execution tree is shown in Fig. 4(a). This paradigm can be mapped to the PG OGM. The differences lie in that not only do we have to account for the actual operator processing, but also for the data movement across peers of the system.

The execution plan of Fig. 4(a) is depicted as an operator graph, following the conventions of the PG OGM, in Fig. 4(b). There, we see that the physical logical operators are captured as PG operators in the OGM. Moreover, two peers cooperate in evaluating the query. A temporary result is moved from one peer to another in order for the computation to complete. Note that local computation may in fact be completely asynchronous, as is implied by the use of an asynchronous join processing operator like symmetric hash join [17]; parts of the evaluation can be performed in parallel and the results asynchronously transferred between nodes for faster response time.

4.2 Query Optimization

The optimization paradigm employed by database systems ever since the seminal work of [12] has been cost-based query optimization. Simply put, the query



(a) A database execution plan

(b) The execution plan in PyraDB

Fig. 4. Mapping from an execution plan to OGM

optimizer iterates over alternative execution plans, estimates the cost of each plan and picks the plan with the minimum estimated cost. The basis for the cost model used has traditionally been the cardinality of the inputs. The logic behind the choice is that what really "hurts" performance in a database system is disk I/O and cardinality is a good metric of how much disk I/O is needed to evaluate a query.

Cost-based query optimization is of course applicable in the case of PyraDB; however, we need to tailor it to the distributed execution environment. First of all, cardinality is not the only metric that captures the cost of an alternative execution plan for the query. Since we are operating in a distributed execution environment, we have to take into account that data will be moved around so the rates [16] at which we can receive the data from the various peers become important. Moreover, as we have mentioned, it could be the case that necessary algorithms will have to be shipped to the peers (the computing resource movement operators of Section 3.2); in this scenario we care for the "raw" size of data movement, whether it is a data resource or a computing resource, rather than the cardinality (which is not even applicable for computing resources). Finally, we have mentioned that part of the PG architecture is locating the relevant sources in the first place. This cost needs to be taken into account when deriving the cost model of the system.

4.3 System Evolution

The system evolves in two ways. The first one is quite straightforward: by the introduction of data resources. As in the vein of a P2P system, the PG architecture allows the introduction of a new data resource and therefore a new queryable source for PyraDB. Once these resources are stored in the system's distributed catalog, they are available for querying by any peer of the system.

Perhaps more surprisingly, the system evolves in another way: by introducing new computing resources. In the PyraDB case, a new computing resource may indeed be a new query evaluation algorithm. For instance, consider the case where a new join processing algorithm becomes available to the system. Its

signature is then known by some of the peers and the operator slowly propagates through the system. Contrast this with the case of a centralized or traditional distributed database system, where the introduction of a new algorithm implies the whole query engine and optimizer being recompiled to take into account the new algorithm. This description is admittedly simplistic, in the sense that even for PyraDB the introduction of a new algorithm will have to take some amount of "restructuring" (*e.g.,* the cost function of the algorithm needs to be propagated and incorporated into optimization decisions). This restructuring, however, is minimal in comparison to a specialized database system that is not executed on top of a flexible architecture like the PG architecture.

5 Related Work

A host of Grid-related projects exists [14]. Computational grids like Condor [10] and Globus [5] aim at providing an operating system-like abstraction to computation. Jobs are submitted to a pool of cooperating computers and the system undertakes the task of employing the entire pool to carry out the computation by taking advantage of idle computers and migrating the process to them. Both systems address the computational grid aspects rather than the data grid ones.

Workflow-based database systems are also relevant, in the sense that they employ a database system for storage and data access. They then use the workflow of a scientific computation as the primitive under which data is retrieved from the database and manipulated. To the best of our knowledge, ZOO [1, 8] is the only such system to make this connection. In contrast to the PG approcach, however, ZOO is not aimed to be used in a distributed computing environment.

Implementation of a database system over the Grid is examined in [11], while over a P2P system in [6, 7]. The PyraDB proposal is different from both Grid and P2P databases since it does not rely on any central catalog (as is the case in [11]) and allows for computing resource identification and movement, something that is not the case in either [6] or [7].

6 Conclusions

We have presented Pyragrid, a system architecture that bridges the gap between P2P systems and the Grid. The novelty of the architecture lies in the classification of system resources into data and computing resources, the same separation that is found in the distinction between computational and data grids. This separation, however, is only conceptual in the Pyragrid architecture. The system employs a distributed hash table to map both types of resources to the peers of a larger system. Handling of processing requests is performed by first decomposing the request in terms of its constituent data and computing resources and locating those resources (much in the same way that a peer-to-peer system locates relevant data). Once the resources are identified, an operator graph is composed that captures all operations that need to be performed so that the processing request is satisfied. The operators of the graph capture the movement

of data and computation from peer to peer, as well as the processing taking place at each peer. As a first step towards assessing the viability of the Pyragrid approach, we presented PyraDB, a prototype database system built on top of the Pyragrid architecture. We showed how query evaluation principles can be mapped to Pyragrid's operator graph model and briefly touched on aspects of query execution, query optimization and system evolution.

Acknowledgements. This work was funded by the DELOS Network of Excellence for Digital Libraries.

References

1. Anastassia Ailamaki *et al.* Scientific Workflow Management by Database Management. In *Proceedings of SSDBM*, 1998.
2. Rob Allan *et al.* Grid Information Systems (Draft). Technical Report UKeS-2003-04, UK National e-Science Center, December 2003.
3. Philip A. Bernstein and Dah-Ming W. Chiu. Using Semi-Joins to Solve Relational Queries. *JACM*, 28(1):25–40, 1981.
4. Burton H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *CACM*, 13(7):422–426, 1970.
5. Ian T. Foster. The Globus Toolkit for Grid Computing. In *Proceedings of CCGRID*, 2001.
6. Matthew Harren *et al.* Complex Queries in DHT-based Peer-to-Peer Networks. In *Proceedings IPTPS*, 2002.
7. Ryan Huebsch *et al.* Querying the Internet with PIER. In *VLDB Conference*, 2003.
8. Yannis E. Ioannidis *et al.* ZOO : A Desktop Experiment Management Environment. In *VLDB Conference*, 1996.
9. Jaewoo Kang, Jeffrey F. Naughton, and Stratis D. Viglas. Evaluating Window Joins over Unbounded Streams. In *ICDE Conference*, 2003.
10. Michael J. Litzkow *et al.* Condor - A Hunter of Idle Workstations. In *Proceedings of ICDCS*, 1988.
11. David T. Liu and Michael J. Franklin. GridDB: A Data-Centric Overlay for Scientific Grids. In *VLDB Conference*, 2004.
12. Patricia G. Selinger *et al.* Access Path Selection in a Relational Database Management System. In *SIGMOD Conference*, 1979.
13. Ion Stoica *et al.* Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *SIGCOMM Conference*, 2001.
14. United Devices. Grid computing projects, 2004. `http://www.grid.org`.
15. Stratis Viglas, Jeffrey F. Naughton, and Josef Burger. Maximizing the Output Rate of Multi-Way Join Queries over Streaming Information Sources. In *VLDB Conference*, 2003.
16. Stratis D. Viglas and Jeffrey F. Naughton. Rate-Based Query Optimization for Streaming Information Sources. In *SIGMOD Conference*, 2002.
17. Annita N. Wilschut and Peter M. G. Apers. Pipelining in Query Execution. In *Conference on Databases, Parallel Architectures and their Applications*, 1991.

Moving Digital Library Service Systems to the Grid

Leonardo Candela, Donatella Castelli, Pasquale Pagano, and Manuele Simi

Istituto di Scienza e Tecnologie dell'Informazione "Alessandro Faedo" - CNR
Via G. Moruzzi, 1 - 56124 PISA - Italy
{candela, castelli, pagano, simi}@isti.cnr.it

Abstract. The architecture of a digital library service system strongly influences its capabilities. In this paper we report our experience with the OpenDLib system, which is based on a service-oriented architecture, and we describe how, in the attempt to better satisfy the user requirements, we decided to develop a digital library service-oriented infrastructure on Grid. We also briefly introduce this infrastructure and present the open issues related to its development.

1 Introduction

Four years ago, the DLib group at ISTI-CNR began to develop a Digital Library Service System (DLSS), i.e. a system for creating and managing digital libraries (DLs). In designing this system, named OpenDLib [2], our aim was to create a customizable system that, when appropriately configured, could satisfy the needs of different application frameworks.

Our first step in designing this system was to clarify what we meant for DLs and to identify the features that a system able to implement DLs should provide. This analysis brought us to understand that a DLSS is a complex system that must not only offer powerful user functionalities (e.g. search, browse, annotation) but it must also implement basic functions for supporting the fruition of the user functionality and for guaranteing the quality of the overall DL service, e.g. its availability, scalability, performance. Moreover, it must satisfy a number of other desiderata, like be extensible, easy to install and to maintain.

In order to create the conditions for achieving the required level of quality we analyzed a range of possible system architectures and, finally, we decided to adopt a distributed, dynamically configurable, service-oriented architecture (SOA). The development of a DL system based on this architecture required a greater implementation effort with respect to the development of centralized system since a number of services dedicated to the co-ordination, management and optimal allocation of the different service instances had also to be provided. OpenDLib is now an operational system that has been used for building a number of DLs [13, 14]. Each of these DLs has its own specific distributed architectural configuration that reflects the needs of the application framework where it operates. In all these different frameworks the distributed service architecture has

proved to be a valid instrument to satisfy a number of requirements that could not have been met otherwise. The greater development effort has thus be highly compensated by the better quality of the DL functionality exposed to the users.

New architectural approaches have emerged, or have been consolidated since we designed the OpenDLib system, e.g. Web services [4, 12], P2P [11], Grids [9, 10]. These approaches provide features that simplify the implementation of a distributed DL architecture and offer a number of new possibilities for implementing novel user functionalities and for enhancing the quality of the overall DL.

The DLib group at ISTI-CNR, with a number of other European research organizations and software companies, have recently set up a project, DILIGENT, for creating a DL infrastructure that exploits these new approaches. In particular, our plan is to built DILIGENT as a service-oriented application of the Grid infrastructure released by the EGEE Project [6].

In this paper we introduce the OpenDLib architecture by focusing on the aspects related to the services management and we report our experience in operating this system. We also describe why we decided to move towards the new DILIGENT architectural framework, what we expect from it, and the open research issues that must be addressed before starting its development.

The rest of this paper is structured as follows: Section 2 introduces the main requirements that motivated our choice of a distributed service architecture for OpenDLib; Section 3 describes the OpenDLib architecture and the lessons learned while developing and using it; Section 4 briefly introduces the DILIGENT DL infrastructure by describing its architecture and highlighting its open issues; finally, Section 5 concludes.

2 Requirements for a digital library system

The objective of the OpenDLib project was to create a software toolkit that could be used to set up a digital library according to the requirements of a given user community by instantiating the software appropriately and then explicitly submitting new documents or harvesting the content from existing sources.

The design of the system was strongly influenced by the requirements collected from some potential user communities. In particular, the requirements that mostly influenced the OpenDLib architectural choices are:

1. There is a set of core DL functionalities, such as search, retrieval, access to information objects, that any DL should provide. The format in which each of these functionalities is presented to the user is usually different since it complies with the application specific vocabularies and rules. In addition to the core functionalities, each DL, usually, *must provide other specific functionalities for serving the application-specific requirements.*
2. New organizations may ask to participate to the DL during its lifetime and additional functionalities may be required to satisfy new needs. *A DL must be able to dynamically evolve by adapting itself to these new situations.*

3. The handling of a DL can be expensive in terms of financial, infrastructural and human resources. Many organizations are confident that this problem can be overcome by adopting a *DL federated model*. According to this model, multiple organizations can set up a DL by sharing their resources without losing, if required, control over their own resources. For example, they can store their information objects locally or host key services on their computers.
4. *The users of a DL require a good quality of the service (QoS)*, i.e. an acceptable level of non-functional properties such as performance, reliability, availability and security.
5. *Access to content and services is usually regulated by policies*. These can specify, for example, that a collection of objects is only visible to a particular group of users, or that a set of services can only be used free of charge for a given time interval.

In order to satisfy the above requirements, we decided to adopt a service-based architecture, i.e. an architecture in which all the functionalities are defined as independent services, with well-defined invokable interfaces, that can be called in defined sequences to form business processes. Following this architectural paradigm, we implemented OpenDLib as a federation of services, each of which implements a well defined functionality.

The next section describes the DL architectural framework that we designed following this choice.

3 The OpenDLib architectural framework

The OpenDLib architectural framework consists of an open and networked federation of cooperating services[1]. These services cooperate, via message exchange, in order to implement the OpenDLib functionality. In particular, the federation comprises a number of services that implement the user functionality. These services, collectively called *application service*, are listed in Table 1.

The communication among services is more complex than a simple client-server application. A service can act both as a provider and as a consumer, and use relationships may exist a priori among any subset of the services. In fact, services can be combined in different ways to support different functionality, and the same services may be used in different ways, in accordance with the restrictions placed on its use and the goal of use.

The communication among services is regulated by the OpenDLib Protocol (OLP) [3]. Such protocol imposes a set of rules that a service must follow in order to communicate with the other services belonging to the federation. OLP requests are expressed as URLs embedded in HTTP requests. All structured requests and responses are XML-based.

[1] Hereafter, with the term "service" we mean an OpenDLib software module that supplies a certain task via a well-defined interface. Each module is able to communicate with other OpenDLib modules and can be deployed and hosted on a server.

Service name	Main performed tasks
Repository	Storage and dissemination of documents that conform to a document model termed DoMDL [1]. These documents can be structured, multilingual and multimedia.
Multimedia Storage	Storage, streaming and downloading of video manifestation of a document, dissemination of videos either as whole documents or as aggregations of scenes, shots and frames.
Library Management	Submission, withdrawal and replacement of documents. It is configurable with respect to the metadata formats accepted.
Index	Document retrieval parametric w.r.t. the metadata format, the set of indexed fields, the result set format and the query terms language.
Query Mediator	Document retrieval by dispatching queries to the appropriate Index service instances and by merging the result sets, taking into account the peculiarities of available Index instances.
Browser	Construction and use of appropriate data structures, termed indexes, for browsing the library content, parametric w.r.t. the metadata formats, the set of browsable fields, and the result set format.
User Interface	Mediations among human user and application services.

Table 1. OpenDLib application services.

The OpenDLib services can be centralized, distributed or replicated on different hosting servers. An OpenDLib DL thus usually comprises multiple instances of the same service type hosted on remote servers of different organizations. This distribution provides an appropriate context for supporting the DL federated organizational model and for ensuring quality attributes such as performance and scalability.

All the OpenDLib services of the current release are highly configurable. It is possible, for example, to select the accepted metadata formats of an Index, the query language of a Query Mediator, the publishing and service hosting institutions, the number of service replica, etc. This provides a great flexibility and permits to use the system in a variety of different DL application frameworks.

OpenDLib supports three kinds of dynamic expansions: 1) new services can be added to the federation; 2) new instances of a replicated or distributed service can be mounted on either an existing or a new hosting server; 3) the configurations of the services can be modified so that they can handle new document types, new metadata formats and support new usages. By exploiting these kind of expansions new application specific needs can be easily satisfied.

The management of a dynamic, customizable, independent set of services aimed at ensuring the desired quality of service is not trivial. It involves many functions such as: *security*, e.g. authorization of the request, encryption and decryption as required, validation, etc.; *deployment*, allowing the service to be redeployed (moved) around the network for performance, redundancy for availability, or other reasons; *logging* for auditing, metering, etc.; *dynamic rerouting*

for fail over or load balancing and *maintenance*, i.e. management of new versions of the service or new services to satisfy new users needs.

In the rest of this section, we focus our attention on these services management aspects presenting the approach that has been taken in OpenDLib. In particular, we describe the two elements that together supply a "basic infrastructural layer" for supporting an optimal co-operation among the distributed services: the *Manager Service*, which maintains a continuously updated status of the networked federation of services, checks its consistency and controls the flow of the communication, and the *OpenDLib kernel* module that implements the rules imposed by the protocol embedded in each application service.

3.1 The Manager Service

We have seen that OpenDLib supports different kinds of "on-the-fly" expansions. Most of these expansions, even when they regard a specific service instance, also require a change in the configuration of other instances in order to allow them to consider the new characteristic of the service. For example, when a Repository instance is modified to accept a new metadata format, at least one Index instance must be updated to index the new format; when a new Query Mediator instance is set up to reduce the workload on the existing Query Mediators, then a certain number of User Interfaces must change their communication flow and address their service requests to the new instance.

Similar updates are needed when the conditions of the underlying network change, e.g. when there is a network failure, when the number of requests sent to a service instance exceeds an established threshold.

All the above updates in the configuration of the federation are controlled by the Manager Service, which following established algorithms, always derives the best routing strategy required to achieve a good QoS.

The Manager service is partially configured by the DL administrator at the start-up of the system. Its configuration parameters contain the minimum information required to specify the DL topology: e.g. the address of the hosting servers; the list of the services and whether they are centralized, replicated or distributed; the number of instances for each service; their allocation to the servers, etc. Configuration parameters contain also a number of consistency rules that specify the legal configurations of the service instances in the federation. These rules strictly depend on the type of service and on the "use" relation that links them. For example, the language of the terms in a query that is processed by the Query Mediator must be one of the languages indexed by the used Index services, the document and metadata descriptions submitted to a Library Management Service must conform to those managed by the corresponding Repository.

By exploiting the information about the DL architecture acquired at start-up time, the Manager begins to collect more detailed information about the service instances by periodically sending them appropriate protocol requests. It then processes the information gathered, controls its consistency, and takes decisions about the organization of the federation, such as, for example, the communication paths among the services instances. These decisions can change over time

according to the value of different parameters, such as the set of running service instances, the workload of a server and the status of the connection.

All service instances notify the Manager of any changes in their configuration and on the status of the service federation. The Manager updates the architectural map and executes the necessary steps to collect information about any new instance. The service instances periodically harvest information about the federation from the Manager. For example, each service that uses another replicated service asks for the address of the instances that can serve its requests.

The instances of the federation can configure themselves either by directly exploiting the information received from the Manager, or by sending appropriate service requests to the instances whose addresses have been obtained through the Manager. Once configured, the various instances can start the co-operation required to process the DL user requests.

3.2 The OpenDLib kernel

When designing and implementing a service-oriented architecture, a common task is to provide an high level abstraction of the communication among the distributed services. This point have been carefully planned in the OpenDlib system.

The OpenDLib kernel is the basic software layer where the OpenDLib services are built on and it includes tools and software modules that allow a service to use the OpenDLib Protocol and to speak to each other no matter what is their physical location.

In order to meet these needs, a small messages dispatcher module is installed on each OpenDLib network node. This module is in charge to dispatch the incoming messages to the appropriate service hosted on the node. When a service sends a request to a remote service, it actually interacts with the dispatcher module of the target node. Afterwards, the request is interpreted and converted in a manner that a generic OpenDLib service is able to manage.

Another main step in this direction is represented by the *ProtocolManager* module. This part of the kernel is responsible to build a service request according with the rules of the protocol. Moreover, it allows to interact both with remote and local services in a transparent way. In order to achieve this second functionality a careful design of the services interface has been conducted. The study produced a solution that facilitate the conversion of a network message in a local one if the producer and the consumer of the message are hosted on the same node.

Finally, the kernel provides a set of utilities that simplify the management of the configuration and status of the federation of services disseminated by the Manager Service.

Major results of these efforts are:

- if the communication needs to be changed for any reason, the change is localized in some specific points only;

– an abstraction layer is provided, so new services can be added more easily, avoiding to deal with communication details;
– a mechanism that permits an optimization of the network traffic among the different nodes is provided.

3.3 Lesson Learned

OpenDLib is now a running system. A number of DLs [13, 14] have been built using it and several others are under construction. The experience made during this experimentation has validated some of our design choices. In particular, we now firmly believe in the power of a service-oriented architecture for DLs. This type of architecture provides an extensible framework where application specific services can be added to satisfy the local requirements. Moreover, by supporting a federated maintenance model, it naturally supports the creation of large DLs obtained by dynamically and progressively aggregating resources provided by many disperse small organizations.

The experience made so far, however, has also highlighted a number of weakness of the DL system we have built. In particular:

– In the four years that have been spent in implementing and experimenting OpenDLib both the enabling technologies and the standards evolved. Today, web services and SOA are highly diffused and quite *standard* solutions for web applications. Many specifications dealing with common issues of this framework, e.g. security, resource description, etc., springs up in the WS-* family. Grid technology, offering also computational and storage power on demand, is now meeting these technologies with the WSRF. P2P technologies, popularized by (music) file sharing and highly parallel computing applications (e.g. SETI@home), can be employed successfully to reach some design goals such as scalability, availability, anonymity. Certainly, these new standards and technologies provide more powerful and advanced solutions than the ad-hoc one originally implemented by OpenDLib.
– Multimedia and multi-type content information objects are emerging as alternative and more powerful communication vehicles. In OpenDLib, the handling of these objects is limited since, e.g. multimedia documents may require high computational capacity to be opportunely and fully managed and employed.
– Many of the user communities that demand for DLs are small, distributed, and dynamic; they use the DL to support temporary activities such as courses, exhibitions, projects. Even if a digital library service system like OpenDLib can be used to setting a DL reducing the cost, these costs are still too high for certain communities of users. Each time a new DL is created appropriate computer and storage resources are needed for hosting the OpenDLib services. Furthermore, specialized technical staff with appropriate skills is required to configure, install and maintain the system.

Certainly, the recent advances of the technology makes now possible to propose better solutions to the requirements presented in Section 2. With this purpose

in mind and in order to overcome the above limitations, our group, with other European research and industrial organizations, has recently set up a new EU FP6 integrated project, DILIGENT, that will start its activity on September 2004. The aim of this project is to design and experiment the development of a new SOA DL infrastructure on a Grid enabled technology. The objectives of this project and the expected results are briefly introduced in the next section.

4 The next step: DILIGENT

The main objective of the DILIGENT project is to create a knowledge infrastructure that will allow members of dynamic user communities to build-up *on-demand transient virtual DLs* (VDLs) that satisfy their needs. These DLs will be created by exploiting shared resources, where by resources we mean content repositories, applications, storage and computing elements.

The DILIGENT infrastructure will be an evolution of a DLSS. By exploiting a wider notion of sharing, this infrastructure has the potential of reducing the cost of setting up DLs thus enabling a larger adoption and use of these systems.

The DILIGENT infrastructure will be constructed by implementing a number of DL services in a Grid framework. In particular, DILIGENT will exploit the efforts of the EGEE project [6] by relying on the Grid infrastructure that will be released at the end of this project. We expect that by merging a service-oriented approach with a Grid technology we will be able to exploit the advantages of both. In particular, the Grid should provide a framework where a better control of the shared resources is possible. Moreover, it should enable the execution of very computational demanding applications, such as those required to process multimedia content.

Our current plan is to develop all the DILIGENT services following the rules established by the new Web Service Resource Framework (WSRF) [5, 7]. This framework, which comprises six Web services specifications defining the *WS-Resource approach* to modelling and managing state in a Web services context, is very suitable to satisfy the needs of a DL application where both stateless and statefull resources cohabit.

The services that will be developed by the DILIGENT infrastructure project can be logically organized into three main layers:

- *Digital Library Layer.* This layer consists of a set of reliable and dependable production-quality services covering the core functionalities required by DL applications. This set provides submission, indexing and discovery of mixed-media objects (documents, videos, images, environmental data, etc.), and the management and processing of these objects through annotation, composition, cooperative editing, etc. It also supports the dynamic creation and access to transient VDLs. Each service of this area will likely represent an enhancement of the functionalities provided by the equivalent non-Grid-aware service as it will be designed to take full advantage of the scalable, secure, and reliable Grid infrastructure.

– *Application-Specific Layer*. This layer contains the set of services provided by users communities that have decided to share their legacy content and application-specific resources.
– *DILIGENT Collective Layer*. This layer is composed by services that enhance existing Grid collective services, i.e. those global services needed to manage interactions among resources, with functionalities able to support the complex services interactions required by the Digital Library Layer.

The services of the Collective Layer (see Table 2) and those provided by the Grid middleware, play the same role of the "basic infrastructural layer" in OpenDLib as they manage the federation of services and resources that implement a VDL. In this environment, however, the management is more complex since the set of servers where the DL services are hosted is not known a priori and it can vary dynamically during the VDL lifetime.

Service name	Main performed tasks
Information Service	Discovering and monitoring of a set of distributed resources, enabling other services to be aware of the environment, or part of it, that hosts them. By maintaining a *real-time* monitoring of the whole set of available resources, single services and VDLs can be enabled for self-tuning resource usage and workload-balancing maximizing the use of available resources.
Broker & Matchmaker	Optimal distribution of services and resources across Grid nodes, by promoting efficient usage of the infrastructure through the identification of the *best* Grid node where to allocate a service or a resource.
Keeper	Bringing together the set of services belonging to a VDL by assuring the QoS characteristics required. Create, address, inspect and manage the lifetime of all the resources needed to satisfy the definition criteria of the library.
Dynamic VO Support	Creation of the Grid *operational context* associating users, user requests and set of resources that enables VDLs to work accordingly with the resource sharing policies and agreements.

Table 2. DILIGENT Collective Layer services.

This new DL architecture infrastructure, at least from the theoretical point of view, provides the support required to meet the requirements that we have identified previously and to overcome the limitations that we have experimented with OpenDLib. However, combining concepts and techniques belonging to different research fields and disciplines (DL, Information Retrieval, Grid, data management, Web Services, information systems, P2P, etc.), the DILIGENT project has to solve at least the following open issues:

– Sharing of resources is acceptable only if it is highly controlled. Strong and valid security and policy mechanisms that take into account the rules of the DL providers and consumers must be developed.

- In order to dynamically create a VDL the system must be able to automatically select and retrieve the resources that better match the demand of the library creator. This requires an appropriate description of the DL resources and powerful discovery mechanisms.
- The SOA approach proposed for the DILIGENT infrastructure defines only the higher level structure of the architecture. Each service belonging to the library can internally adopt a different architecture, e.g. an Index can be realized both using a P2P technique as well as a centralized service. This heterogeneity certainly complicates the digital library management and demands for more sophisticated algorithms.
- The different architectural framework in many cases may suggests better algorithms for implementing the DL functionality. For example, a Grid framework, which offers a high computing capacity, may enable the exploitation of complex but powerful algorithms which were not realistically possible in the traditional DL frameworks. One of the challenges in DILIGENT will certainly be to identify new algorithms that exploit the capabilities of this new DL framework.

5 Conclusion

The paper describe the architecture of the OpenDLib DL service system and it introduce the main architectural plans for the DILIGENT knowledge infrastructure.

In carrying out this experience we have learnt that the realization of a DL system not only requires the implementation of the functionality that are directly perceived by the users but it also requires the development of what we called a "basic infrastructural layer". In this paper we mainly discuss this layer from the point of view of the management of the DL services, but there are other important functionalities that a basic infrastructure layer should provide, like independence from the physical organization of the content, policy control, etc. The extent of the basic infrastructure layer strongly depends on the underlying architecture choices. The more distributed, flexible, dynamic, customizable the architecture framework is, the greater is the layer required to ensure the quality of the DL services.

In OpenDLib this layer has been constructed from scratch, by adopting ad-hoc solutions. At the present, however, more standards and powerful solutions are possible. With the DILIGENT project, our challenge is to build a new DL service system that takes into account these new openings and to contribute to the evolution of the architectural infrastructure for future DLs.

References

1. Donatella Castelli and Pasquale Pagano. A flexible Repository Service: the OpenDLib solution. In J. Á. Carvalho, Arved Hübler, and Anna A. Baptista, editors, *Proc. of the* 6^{th} *International ICCC/IFIP Conference on Electronic Publishing*, pages 194–202, 2002.

2. Donatella Castelli and Pasquale Pagano. OpenDLib: A Digital Library Service System. In *Proceedings of the* 6^{th} *European Conference on Digital Libraries (ECDL2002)*, pages 292–308. Springer-Verlag, 2002.
3. Donatella Castelli and Pasquale Pagano. The OpenDLib Protocol. Technical report, Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", CNR, 2004.
4. Ethan Cerami. *Web Services Essentials (O'Reilly XML)*. O'Reilly & Associates, 2002.
5. Karl Czajkowski, Donald F. Ferguson, Ian Foster, Jeffrey Frey, Steve Graham, Igor Sedukhin, David Snelling, Steve Tuecke, and William Vambenepe. The WS-Resource Framework. *White paper*, 2004.
6. EGEE Team. EGEE: Enabling Grids for E-science in Europe. http://public.eu-egee.org.
7. Ian Foster, Jeffrey Frey, Steve Graham, Steve Tuecke, Karl Czajkowski, Donald F. Ferguson, Frank Leymann, Martin Nally, Tony Storey, William Vambenepe, and Sanjiva Weerawarana. Modeling Stateful Resources with Web Services. *White paper*, 2004.
8. Ian Foster and Adriana Iamnitchi. On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. In M. Frans Kaashoek and Ion Stoica, editors, *Peer-to-Peer Systems II, Second International Workshop, IPTPS 2003, Revised Papers*, volume 2735, pages 118–128, 2003.
9. Ian Foster, Carl Kesselman, Jeffrey Nick, and Steve Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum, June 2002.
10. Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the Grid: Enabling scalable virtual organization. *The International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.
11. Nelson Minar, Marc Hedlund, Clay Shirky, Tim O'Reilly, Dan Bricklin, David Anderson, Jeremie Miller, Adam Langley, Gene Kan, Alan Brown, Marc Waldman, Lorrie Cranor, Aviel Rubin, Roger Dingledine, Michael Freedman, David Molnar, Rael Dornfest, Dan Brickley, Theodore Hong, Richard Lethin, Jon Udell, Nimisha Asthagiri, Walter Tuvell, and Brandon Wiley. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly & Associates, 2001.
12. Eric Newcomer. *Understanding Web Services: XML, WSDL, SOAP, and UDDI*. Addison-Wesley, 2002.
13. OpenDLib Team. e-Library: a public OpenDLib instance. http://elibrary.isti.cnr.it.
14. OpenDLib Team. The ARTE Library: an OpenDLib instance. http://odl-server1.isti.cnr.it.

Hyperdatabase Infrastructure for Management and Search of Multimedia Collections

Michael Mlivoncic, Christoph Schuler, and Can Türker

Swiss Federal Institute of Technology Zurich
Institute of Information Systems, ETH Zentrum
CH–8092 Zurich, Switzerland
{mlivonci|schuler|tuerker}@inf.ethz.ch

Abstract. Nowadays, digital libraries are inherently dispersed over several peers of a steadily increasing network. Dedicated peers may provide specialized, computationally expensive services such as image similarity search. Usually, the peers of such a network are uncoordinated in the sense that their content and services are not linked together. Nevertheless, users expect to transparently access and modify all the (multimedia) content anytime from anywhere not only in an efficient and effective but also consistent way. To match these demands, future digital libraries require an infrastructure that combines various information technologies like (mobile) databases, service-oriented architectures, peer-to-peer and grid computing. In this paper, we sketch such an infrastructure and illustrate how an example digital library application can work atop it.

1 Introduction

Future digital libraries shall provide access to any multimedia content anytime and anywhere in a user-friendly, efficient, and effective way. One problem of nowadays digital libraries is that they are dispersed over a network in an uncoordinated fashion such that each peer of the network often works isolated without regarding other peers that manage related content. Another problem is to efficiently handle the steadily increasing amount of requests for multimedia content. Besides, the distributed content should be kept consistent and provided in a personalized way. Hence, an infrastructure is required for digital libraries that is a reliable, scalable, customizable, and integrated environment.

To build such an infrastructure, the best aspects of (mobile) databases, service-orientation, peer-to-peer and grid computing must be combined. We refer to such an infrastructure as a *hyperdatabase* [1]. Within such an environment, databases and special servers provide basic services, such as efficient and reliable storage for various kinds of multimedia data like image, audio, and video. Service-orientation [2] helps to describe, customize, and deploy complex services, such as sophisticated search of images with respect to their content and annotations. The peer-to-peer paradigm [3] allows a loosely-coupled integration of digital library services and ad-hoc sharing of information, such as recommendations and annotations. Since certain services within a digital library are computationally intensive, e.g., the extraction of features from multimedia objects to

support content-based similarity search, grid technology [4] supports the optimal utilization of the given computing resources.

In the recent years, we have developed OSIRIS (**O**pen **S**ervice **I**nfrastructure for **R**eliable and **I**ntegrated process **S**upport) [5], which is a prototype of a hyperdatabase infrastructure. As common standards like WSDL and SOAP, OSIRIS helps to access a wide range of service types and allows isolated service calls. In addition, OSIRIS support processes (compound services) as a means for combining existing services and executing them under certain (transactional) guarantees. In a digital library, such services can be used, for instance, for the maintenance of dependencies among the various data repositories of the digital library. In this way, a sophisticated search may benefit from always up-to-date indexes.

In this paper, we describe how to organize such a self-contained application. The ISIS (**I**nteractive **SI**milarity **S**earch) application demonstrates an efficient management and organization of large multimedia collections atop OSIRIS. It features a wide range of functions that allow for metadata management as well as efficient and effective content-based similarity search on multimedia objects. The implementation of ISIS consequently follows the idea of service-orientation. Specifically, all digital library functionality is encapsulated by services, such as storage services for arbitrary types of media objects or feature extraction services for given media types. Such services are used to compose the entire ISIS application.

The infrastructure monitors the service execution as well as the routing of the corresponding services. Whenever necessary, it distributes and replicates certain services on multiple grid peers in order to boost the performance.

In principle, any service can be executed locally in a stand-alone manner, as with mobile devices, for example. However, due to several reasons, e.g., resource or license restrictions, some services might be locally unavailable. In such cases, the service execution depends on external services, which have to be bound and invoked on demand. On the other hand, in some cases the locally available services are sufficient to completely execute a compound service. For instance, even a content-based image similarity search does not require a feature extraction service provided the reference image is already part of the collection (i.e., is already indexed).

It is important to note that the entire service execution is transparent to the service designer. She therefore can fully focus on specifying and implementing the core service functionality itself. Service runtime environment tasks (startup, updates,...) and communication with the outside world are handled by the hyperdatabase infrastructure. As first experimental evaluations show, this infrastructure does not only support dynamic changes of the execution environment but is also able to scale with the expected huge number of users, services, and grid peers of future digital libraries.

The rest of this paper is organized as follows: Section 2 gives an overview of ISIS together with the requirements for the underlying infrastructure. Section 3 describes such an infrastructure, OSIRIS, and discuss various important issues like service execution and service registration. Finally, Section 4 concludes.

2 ISIS – A Service-Oriented Application

ISIS (Interactive SImilarity Search) is a powerful service-oriented application for efficient management and organization of multimedia collections. The application features a wide range of functions that allow meta data management as well as efficient and effective content-based similarity search on multimedia objects. The realization of ISIS consequently follows the idea of service-orientation. All basic functionality is encapsulated by a set of services.

We distinguish five classes of such services:

Storage Services. These services provide storage for arbitrary types of media objects. A storage service at a certain peer may be dedicated to a certain content like video clips or images. Storage services can also act as web-caches for remote content in order to speed-up access to often used data. A storage service closely monitors its attached repositories for changes. If, for example, a new object is added to one of its repositories, it will issue a "new object" event. Likewise, a (local) deletion of an object will lead to an "object-deleted" event. These events can be handled by the digital library infrastructure, for example, to keep the consistency between object data and indexed information.

Metadata Services. Such services maintain keyword annotations, textual descriptions, as well as other object properties, such as the membership in several (sub)collections, or predicates like "copyrighted". As an media object might be available in different versions at different locations (e.g, as thumbnail of the image at thumbnail-storage, primary high-resolution image copy at remote storage location), metadata services also keep track of those locations and corresponding properties of the object versions at the various locations (e.g. resolution or thumbnail/primary-role). There is no fixed vocabulary for describing such properties. Hence, ISIS can store arbitrary information about the objects. For example, one might want to store textual information gathered from the surrounding web pages together with an image or the artist and the song title together with a piece of music. Besides such "per-object" information, metadata services also maintain general knowledge about the object types themselves. This includes existing feature descriptors for an object type, the availability of feature extractors and indexing containers within the digital library infrastructure that are able to manage a given feature descriptor.

Feature Extraction Services. Content-based retrieval depends on features that describe the raw content of media objects in a certain feature domain. For example, the pixel information of an image can be described in the color or texture feature domain [6–8], while a piece of music can be described in terms of beat and pitch [9]. A feature is therefore a kind of descriptive fingerprint for an object. In content-based retrieval, object similarity is expressed as similarity of their descriptors in a given feature domain. For a given media type, there could be several feature extraction services computing various kinds of feature descriptors. Also, there is more than one way to

describe a concept like color. For example, we could use a histogram with 64, 256 or whatsoever bins. We could also use color moments. There are many meaningful descriptors and variants around. One might want to use several of them — even in combination. In many cases, the extraction of features is a computationally very expensive task. Therefore, feature extraction can profit from grid infrastructures.

Indexing and Search Services. Indexes allow for efficient search of objects. Different indexes have to handle data from different domains: Besides often high dimensional feature descriptors, object predicates and numerical values as well as keyword annotations of the objects should be indexed efficiently. Searching over an arbitrary *combination* of those attributes efficiently is a non-trivial task. Efficient search strategies on this level are however beyond the scope of this paper. For further information please refer to [10–12]. At this point, we only state that concept and design of such a service should be carefully chosen in a way that it allows for massive scalability through dynamic partitioning of a query onto a set of several search services. Those services will be spread all over the grid infrastructure, similar to the extraction services.

Presentation Services. These services provide frontend functionality to browse and query the multimedia collection and also to initiate some maintenance and administrative tasks. In ISIS, this task is shared among services for the interactive part (frontend), the layout part (XML-to-HTML rendering) and supporting services (session management and template repository).

The entire ISIS application consists of a set of compound services over these basic services. Once the application logic is divided into such basic services, they can be distributed and replicated without changing the description (implementation) of any (compound) service. Figure 1 shows the insertion of a multimedia object as an example for a compound service. The given service is triggered by the "new object" event of a storage service mentioned above when a new media object physically enters the repository.

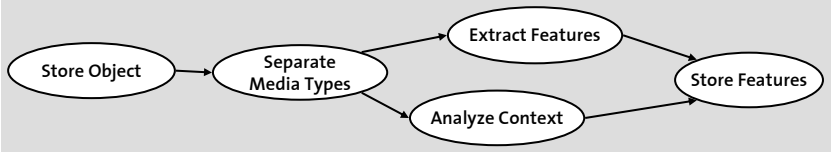


Fig. 1. Compound Service Insert Multimedia Object

The first activity of the compound service is to store the object, i.e., the location and available meta information about the object is stored by the metadata service. Depending on the media type, further information is extracted. In case of a web document, the object will not only contain an image, but also some text surrounding this image on the page. By analyzing the HTML source, it is

possible to gather some layout information and applying some heuristics in order to determine textual descriptions that might be relevant for the image. This text can be indexed later on. Independent of the analysis of the context of the image and its surrounding, the extract features activity uses the raw pixel information of the image to extract several descriptors for color and texture. Note that it is transparent to the user, whether this activity is a single service or a compound service which is composed of single feature extraction services. The distinction between a single and a compound service is important for the infrastructure. By explicitly knowing about the semantics of the various activities of a compound service, the infrastructure is able to further parallelize the extraction to achieve better performance. The store features activity hands all gathered object descriptors and metadata information over to the metadata service, which will in turn care for the indexing and replication of each data item in a suitable way.

Infrastructure Requirements

ISIS performs some complex and sometimes computational expensive tasks. As ISIS may change over time, e.g., when new feature descriptors are becoming available, we need an infrastructure that provides us with flexibility in order to define and modify the logic of the digital library applications, i.e., the corresponding compound services, as necessary. In order to focus on the core digital library tasks, the infrastructure should support a transparent way of communication among services. For example, while designing the extraction service, we do not want to *explicitly* deal with workload distribution, it should be sufficient to "solicit" that a certain task, e.g., a feature extraction, should be executed on any one of the potential service providers. Thus, the infrastructure must also support service discovery. Service providers should be able to declare what kind of functionality they are offering and service users should be able to find them.

As mentioned before, it can be useful, if application logic specifies compound services over the basic services (cf. the feature extraction example). In such cases, feature extraction as well as search can profit even from a higher degree of parallelism. As multimedia content tends to be storage intensive — especially with large-scale general purpose digital libraries — one might also want to distribute the storage services and searching facilities over the peers of the grid. Besides this complexity, we still demand that the overall digital library should be reliable, i.e. (temporary) unavailability of single service *instances* shall not jeopardize the operation of the overall system. Services once invoked (like the insertion of an object) should lead to a guaranteed execution of the defined activities and thus ensure the consistency of the data within the metadata and indexing services. Mentioning consistency, we also want to ensure highest possible "freshness", i.e., changes should be propagated "immediately" instead of monthly updates like in Google and other major web search engines. On object deletion, references to that object should be removed immediately from all indexes. Also, changes of feature data should immediately be propagated to *any* indexing service related to that data.

3 Hyperdatabase Infrastructure for ISIS

In the following, we sketch how a hyperdatabase infrastructure provides us with all the required functionality as elaborated so far.

3.1 Overview of the Infrastructure

A hyperdatabase supports applications following the ideas of a service-oriented architecture. Using a distributed peer-to-peer network, service requests are transparently routed along the connected peers. Following the concept of service-oriented architecture, a service can be *atomic* or *compound*. Compound services are composed of existing services. The composition is realized using the notion of a transactional process [13]. By defining a data and control flow for processes, the involved service calls must appear in the specified application specific invocation order. Transactional processes allows for providing execution guarantees similar to transactions in classical databases. While databases focus on basic data querying and manipulation operations, a hyperdatabase orchestrates service calls. These service calls are executed according the description of the compound service.

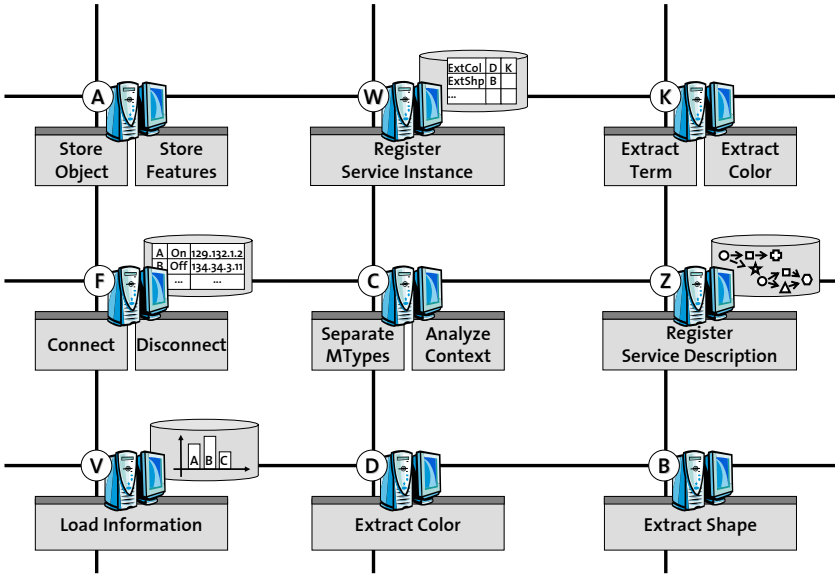


Fig. 2. Hyperdatabase Infrastructure

In order to provide this functionality, the hyperdatabase infrastructure consists of a software layer installed on each peer of the grid (dark gray layers in Figure 2). This layer integrates the peer into the overall hyperdatabase network.

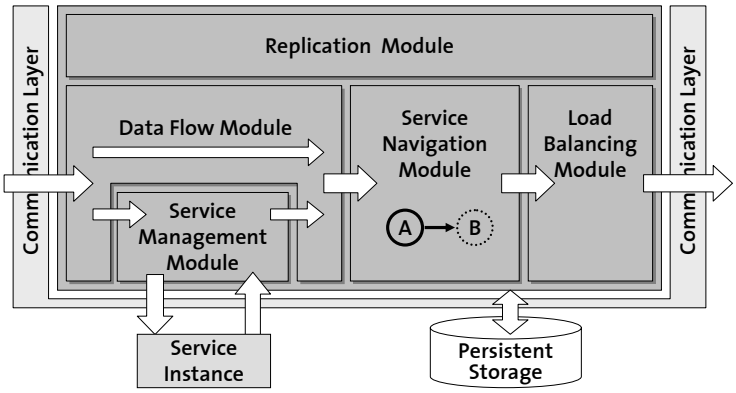


Fig. 3. Hyperdatabase Layer

Ideally, this layer comes together with the operating system like TCP/IP stack does (comparable to the .NET framework). By this integration, local available services can be registered and used by all peers in the grid. Moreover, the local hyperdatabase layer is now enabled to transparently call remote services.

Figure 3 shows the architecture of the hyperdatabase layer, which we will discuss in more detail later. Beside communication, the layer also supports management of compound services and load balancing. This functionality depends on grid common knowledge accessible via the replication manager available at every peer.

OSIRIS [14] as an implementation of a hyperdatabase is realized as a service-oriented application itself. Besides the core hyperdatabase layer, additional system functionality is needed to organize the grid and to provide service transparency and the execution of compound services. OSIRIS implements this functionality also in terms of services. As depicted in Figure 2, OSIRIS provides core services like Register Service Description, Register Service Instance, Connect, and Disconnect. The figure also shows that there is no distinction between core services and atomic application-specific services such as Extract Color or Extract Term.

3.2 Peer-to-Peer Service Execution

As mentioned before, OSIRIS distinguishes two classes of services: atomic versus compound services. The execution of an atomic service consists of one call to a function provided by a service in the grid. A call to a compound service however initiates a peer-to-peer execution. According to the description of a compound service, the contained services, i.e., the process activities, are executed sequentially. Although the execution semantics of the two service classes are different, the invocations of both services are not distinguishable to the caller. In other words, the caller does not care about whether the service is atomic or compound.

The same holds for activities of a compound service, which can be compound services themselves.

OSIRIS provides a transparent way of calling services. A service request that is sent to any peer of the network is transparently routed by the local hyperdatabase layer to an available instance of this service type. For that, OSIRIS implements a distributed service bus that provides routing and load balancing over all existing service instances.

Albeit the infrastructure hides the difference between an atomic and a compound service at the call interface, the execution within the infrastructure layer respects and exploits the differences between atomic and compound services. While the first has to be routed to a service instance, the latter must be driven by the infrastructure itself since its activities can spread over multiple peers of the grid. Therefore, each peer has to provide a minimal service manager that supports a peer-to-peer service execution. This includes the ability to call local services, to navigate to the subsequent activity of the compound service, and to handle execution failures. A compound service consists of a set of ordered services. These services has to be executed according the order defined by the service description. This information is available at the local replication manager. After instantiating a new compound service at any peer in the grid, it is migrated to the hyperdatabase layer of a peer that provides an instance for the first activity.

Figure 3 shows the flow of the service execution of one single step in a compound service. After the service has migrated to the hyperdatabase layer, it enters the *data flow module*. A part of the service data is needed to prepare the call of the service. The *service management module* executes a function at the service instance. After the execution of the service the resulting data is incorporated into the context of the compound service. The next task is to determine the service that has to be executed subsequently. Based on the locally replicated part of the service description, the *service navigation module* decides which service type to call next. The *load balancing module* finally routes the service instance to an available service instance considering system workload. This routing requires grid configuration information. Therefore, the *replication module* has also to provide grid configuration information. During the execution of a compound service, the service instance migrates from one peer to the next in a truly peer-to-peer fashion. After executing the last activity of the compound service, the response is sent back to the caller, as in the case of an atomic service.

Example 1. Assume a multimedia object is inserted into the ISIS digital library by calling our example compound service Insert Multimedia Object. OSIRIS initiates a service instance for this particular service call. After initialization, the instance is routed to the provider of a service for the first activity. In this way, the service instance reaches peer A in our example in Figure 2. In the context of the compound service, then a local call to the service Store Object is performed. After this call, the service instance migrates to peer C in order to execute the service Separate Media Types. This migration is done directly using a peer-to-peer connection between A and C. The next two service calls can be performed

parallel. This requires that the service instance splits and navigates separately along the definition paths. The first part of the instance stays at peer C in order to execute the service Analyze Context, while the second part is migrated to peer K or D. Considering the current workload of the peers D and K, the service instance migrates to peer K. Finally, the service Store Features has to be executed on peer A. Since the original service instance was split and distributed on the network, OSIRIS has to synchronize and join these parts before migrating to peer A. After finishing the local call, the compound service is completely executed and the response is sent to the caller.

3.3 Registration of Service Descriptions

In the previous discussion, we have already distinguished between service types and service instances without explicitly mentioning this. In service-oriented architectures, this distinction enables a transparent routing during service execution. In fact, a service type is called. The infrastructure matches a currently available service instances to perform the execution of the requested service type. To enable this behavior — in the literature referred as *enterprise service bus* [15] — all available service types have to be registered in the system. While calls to atomic services just have to be routed through the system, compound services have to be executed by the infrastructure itself as described in the previous subsection. For that, the description of compound services has to be published to the infrastructure. In OSIRIS, this publication is handled by the service Register Service Description.

Whenever a service description is published using this service, it is analyzed and prepared for replication along the peers. Keep in mind that service execution is done in a peer-to-peer fashion relying on locally replicated information. To provide exactly the information that is required to perform a completely peer-to-peer execution, the service description has to be enriched and divided into parts containing all information concerning the execution of one contained service. These parts of service description is then replicated to the peers hosting an instance of the corresponding service type. This strategy allows peer-to-peer execution to have most information already available at the local replication module.

Example 2. Assume that the description of the compound service Insert Multimedia Object is inserted into the system. The service Register Service Description splits the definition into five parts — one part for each activity of the compound service. The part concerning the service Extract Features contains all information about the parameter handling the call of that service. This information is needed by the *service management module* of the hyperdatabase layer at the peers D, K and B, respectively. In addition, the service navigation module needs to know about the subsequent services. All this information is replicated to the peers D and K immediately after inserting the service description. The peer-to-peer service execution will route every instance of the service Insert Multimedia Object to one of these peers. Consequently, the corresponding part of the service description should be replicated there.

3.4 Registration of Service Instances

A grid infrastructure has to deal with continues changes of the overall system configuration. Service instances may join and leave the system quite frequently. Therefore the infrastructure has to keep track of the current configuration. OSIRIS provides completely transparent service calls by dynamically replicating information about the available service instances to the corresponding grid peers. This replication is based on a publish-subscribe mechanism, which is described in [14]. While the call of a service can occur at any peer, no prior replication can be performed in order to speed up this routing. However, a temporary replication is reasonable since the probability that this information will be needed in the near future is rather high. Beside this temporary replication, information needed during execution of compound services can be predicted by analyzing the service descriptions. In addition, the information about the subsequent services are needed at a particular peer.

Example 3. In the configuration depicted in Figure 3, the peer A holds the replicated information about the current instances of service Separate Media Types since within the compound service Insert Multimedia Object this service must be invoked after the service Store Object, which is provided at peer A. This way a lot of information is replicated along the grid in order to perform and optimize peer-to-peer execution of services. Assume a new instance of the service Separate Media Types is started at peer B and registered via the service Register Service Instance. As a consequence, OSIRIS triggers the replication of additional configuration information to peer B — mainly the part Separate Media Types of the compound service Insert Multimedia Object and information on available instances of Analyze Context. Extract Features is a compound service and thus has no location associated with. Since the (parallel) activities of this compound service are Extract Color and Extract Shape, the location information about these two services are replicated to peer B.

3.5 Stand-alone Service Execution

The peer-to-peer execution of compound services relies on the transparent routing of service calls. OSIRIS can provide this transparent routing only within the same OSIRIS cell. A cell corresponds to a separate grid environment. Within such a cell, each service of the registered peers is provided to all peers of that cell. As a special case, a complete cell can be installed on one peer. This installation includes OSIRIS core services as well as all services of an application such as ISIS. This peer can also be a mobile device. Albeit local services can be used to realize a stand-alone application, some service types like Extract Features cannot be performed efficiently by the mobile peer. In order to execute such a service, the peer should join a larger cell to perform this service on a peer having more resources. We also allow peers to join a cell without registering its available services to the cell. In this case, the peer acts as a service user. This kinds of a join can be performed by using a proxy service, which provides a bridge between

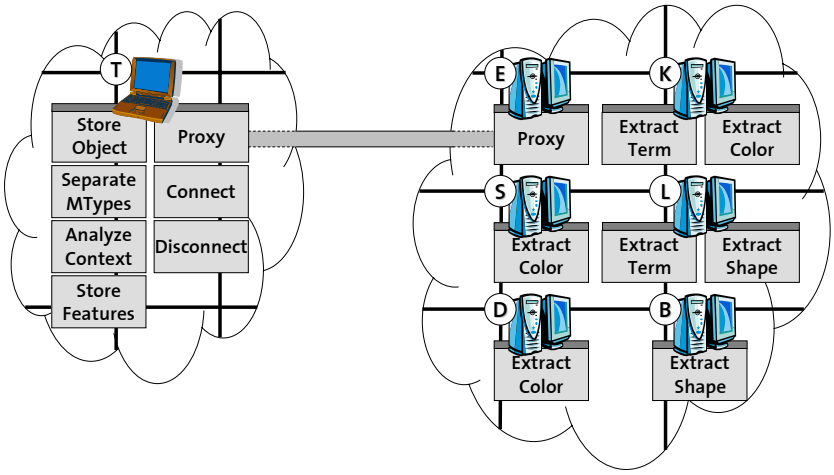


Fig. 4. Proxy Services for Stand-alone Service User

the two cells. As depicted in Figure 4, two instances of the proxy service can be linked in order to establish a bridge. A proxy service provides a service stub and forwards all service calls to the other cell. OSIRIS routes the call to a real service instance.

Note that this forwarding of service calls works also for compound services. However, the peer-to-peer service execution cannot use the bridge to migrate. If the mobile device calls a compound service on the remote cell the proxy service will forward the service call and the complete service will be executed at the remote cell. If a local compound service contains an activity for which only a remote service instance is available, the local proxy service provides a stub for that service. In the context of the local grid, the proxy service executes the service. Therefore, the compound service instance will stay at the local device while executing the service at the proxy service. In this way, the concept of a proxy allows for exploiting services of a remote grid cell.

4 Conclusions

As we have seen in this paper, service-orientation helps to describe and implement complex digital library applications like ISIS as compositions of services. A hyperdatabase infrastructure like OSIRIS, which combines service-orientation with peer-to-peer and grid computing, is then able to exploit the knowledge about the services and their composition to execute them in an optimal fashion. Following the idea of grid computing, the execution of service calls respects parameters like the workload of the peers to dynamically select the best fitting service instance. Besides, OSIRIS is able to replicate service instances on demand on any peer of the grid, and thus to optimize the utilization of the given

resources. The peer-to-peer style of service navigation avoids that the navigation becomes a bottleneck of the overall system, and thus provides the basis for a scalable infrastructure. The current implementation of ISIS atop OSIRIS demonstrates this very nicely.

References

1. Schek, H.J., Schuldt, H., Schuler, C., Weber, R.: Infrastructure for Information Spaces. In: Advances in Databases and Information Systems, Proc. of the 6th East-European Symposium, ADBIS'2002. Volume 2435 of Lecture Notes in Computer Science, Springer-Verlag (2002) 23–36
2. Schmid, M., Leymann, F., Roller, D.: Web Services and Business Process Management. IBM Systems Journal **41** (2002) 198–211
3. Curley, M.G.: Peer-to-Peer Computing Enabled Collaboration. In: Proc. of the Int. Conf. on Computational Science, ICCS 2002. (2002) 646–654
4. Foster, I., Kesselmann, C., Nick, J., Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. In: Global Grid Forum, 2002. (2002) http://www.gridforum.org/ogsi-wg.
5. ETH Database Research Group: OSIRIS: an Open Services Infrastructure for Reliable and Integrated process Support. http://www.osiris.ethz.ch (2004)
6. Niblack, W., Barber, R., Equitz, W., Flickner, M., Glasman, E.H., Petkovic, D., Yanker, P., Faloutsos, C., Taubin, G.: The QBIC Project: Querying Images by Content, using Color, Texture, and Shape. In Storage and Retrieval for Image and Video Databases. Volume 1908 of SPIE Proceedings (1993) 173–187
7. Stricker, M.A., Orengo, M.: Similarity of Color Images. In: Storage and Retrieval for Image and Video Databases. Volume 2420 of SPIE Proceedings (1995) 381–392
8. Dimai, A.: Spatial encoding using differences of global features. In: Storage and Retrieval for Image and Video Databases. Volume 3022 of SPIE Proceedings (1997) 352–360
9. Tzanetakis, G., Cook, P.: Audio Information Retrieval (Air) Tools. In: Proc. Int. Symposium for Audio Information Retrieval, ISMIR 2000 (2000)
10. Weber, R., Schek, H.J., Blott, S.: A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In: Proc. of the 24th Int. Conf. on Very Large Data Bases, VLDB'98, Morgan Kaufmann Publishers (1998) 194–205
11. Weber, R., Schek, H.J., Bollinger, J., Gross, T.: Architecture of a Networked Image Search and Retrieval System. In: Proc. of the 8th ACM CIKM Int. Conf. on Information and Knowledge Management, ACM Press (1999) 430–441
12. Böhm, K., Mlivoncic, M., Schek, H.J., Weber, R.: Fast Evaluation Techniques for Complex Similarity Queries. In: Proc. of the 27th Int. Conf. on Very Large Data Bases, VLDB 2001, Morgan Kaufmann Publishers (2001)
13. Schuldt, H., Alonso, G., Beeri, C., Schek, H.J.: Atomicity and Isolation in Transactional Processes. ACM Transactions on Database Systems **27** (2002) 63–116
14. Schuler, C., Weber, R., Schuldt, H., Schek, H.J.: Peer-to-Peer Process Execution with OSIRIS. In: Proc. of the 1st Int. Conf. on Service-Oriented Computing, ICSOC 2003. Volume 2910 of Lecture Notes in Computer Science, Springer-Verlag (2003) 483–498
15. Chappell, D.: Enterprise Service Bus. O'Reilly (2004)

Data Stream Management and Digital Library Processes on Top of a Hyperdatabase and Grid Infrastructure

Manfred Wurz, Gert Brettlecker, and Heiko Schuldt

University for Health Sciences, Medical Informatics and Technology
Innrain 98 A–6020 Innsbruck Austria
[manfred.wurz|gert.brettlecker|heiko.schuldt@umit.at]

Abstract. Digital libraries in healthcare are hosting an inherently large collection of digital information. Especially in medical digital libraries, this information needs to be analyzed and processed in a timely manner. Sensor data streams, for instance, providing continuous information on patients have to be processed on-line in order to detect critical situations. This is done by combining existing services and operators into streaming processes. Since the individual processing steps are quite complex, it is important to efficiently make use of the resources in a distributed system by parallelizing operators and services. Grid infrastructures already support the efficient routing and distribution of service requests. In this paper, we present a novel information management infrastructure based on a hyperdatabase system that combines the process-based composition of services and operators needed for sensor data stream processing with advanced Grid features.

1 Introduction

Digital libraries in healthcare are increasingly hosting an inherently large and heterogeneous collection of digital information, like electronic journals, images, audios, videos, biosignals, three dimensional models, gene sequences, protein sequences, and even health records. Medical digital libraries therefore have to organize repositories managing this medical information [1] and to provide effective and efficient access to it. In addition, a central aspect is the collection, aggregation, and analysis of relevant information.

Due to the proliferation of sensor technology, the amount of continuously produced information (e.g., biosignals or videos) in medical digital libraries will significantly grow. These data streams need sophisticated processing support in order to guarantee that medically relevant information can be extracted and derived for further storage, but also for the on-line detection of critical situations. Biosignals, like a ECG recording, contain relevant information derived from the evaluation of characteristic parameters, e.g., the heart rate, and their deviance from average. In some cases, even the combination of different biosignals is needed for the extraction of relevant information, such as a comparison

of heart rate and blood pressure. *Data stream management* (DSM) addresses the continuous process streaming data in real-time. Due to the streaming origin of parts of the information stored in medical digital libraries, the latter will significantly benefit from infrastructures incorporating DSM.

Due the service-orientation and the distributed nature of digital libraries (i.e., information is made available by means of services), *Grid infrastructures* are very well suited as basis for digital library applications. The composition of services and DSM operations can be realized by means of processes. The Grid then supports the efficient routing of service requests among different service providers. A very challenging aspect in process-based service composition on top of a Grid environment is that processes itself can be seen as services and therefore can be used within other processes again. This, in a way, adds recursive nature to processes and implements the well known composite pattern [2] for processes on the Grid. Moreover, also the runtime support for process execution can be considered as a special, inherently distributed Grid service.

In this paper, we introduce an integrated hyperdatabase and grid infrastructure that supports the processing of continuous data streams and that is able to distribute the processing of computationally expensive services within a Grid. By this, the requirements of efficiently processing continuous data that can be found in digital medical library applications can be seamlessly supported.

The paper is structured as follows. Section 2 describes a sample telemonitoring application in a digital healthcare library to motivate the need for a joint hyperdatabase and grid environment. In Section 3, we present a process-based approach to data stream management. The dynamic process parallelization by using Grid concepts is introduced in Section 4. Section 5 discusses related work and Section 6 concludes.

2 A Sample Application in a Digital Healthcare Library

In this section, we introduce a sample healthcare application to motivate the need for a flexible and reliable information management infrastructure that supports process management, data stream processing and management, and that provides Grid computing capabilities.

The left hand side of figure 1 illustrates a *telemonitoring system* which takes care of elderly patients suffering from chronic diseases (e.g., diabetes, heart diseases, or other age related problems like Alzheimer). This telemonitoring system is one of the information providers of the underlying medical digital libraries. Patients are equipped with an array of sensors, as for example the LifeShirt-System [3], that continuously measure the patient's body signals (e.g., ECG). Additionally, sensors integrated in the patient's home are detecting context information that describes what the patient is currently doing (e.g., if the patient is sleeping). This information is important to evaluate the medical meaning of vital signs — for example, the ECG signal has to be interpreted differently when a person is sleeping, compared to the case where she is active. In addition to medical monitoring, context information is also used to integrate a patient support system

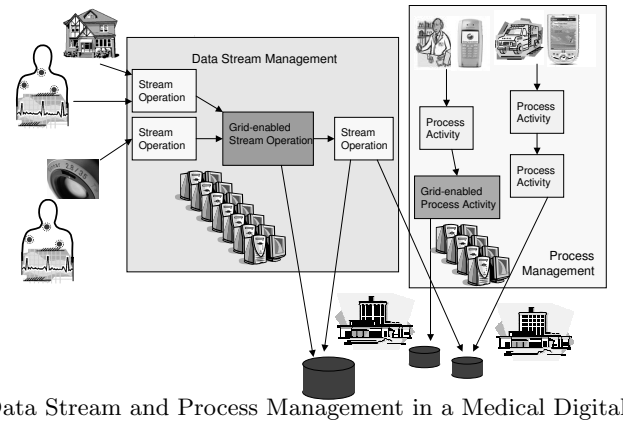


Fig. 1. Data Stream and Process Management in a Medical Digital Library

in this scenario. Patients can be remembered to turn off the oven or take their pills. In order to make use of the vast amount of sensor information, the incoming sensory data has to be processed in real-time. Medically relevant results may be stored in a digital library containing the patient's health record. Results with unknown characteristics are stored in repositories to support medical research. Critical results may request immediate intervention by the caregiver. In this case, appropriate processes (e.g., calling the emergency service or contacting a physician) have to be triggered.

Access to the contents of a medical digital library is supported by special services and user defined processes that combine several of these services (illustrated on the right hand side of figure 1). As described above, processes for contacting the caregiver (e.g., by sending a SMS to a mobile device of a physician), or even for triggering some rescue activities in case of critical situations have to be invoked if necessary. If the physician needs more detailed information or wants to request data on previous treatments or prescriptions, he has to be served with the data in a timely fashion. For all these purposes, appropriate processes have to be available (or have to be defined) and to be executed efficiently by the underlying infrastructure.

Our infrastructure for telemonitoring applications is based on a combined hyperdatabase system [4] and Grid environment [5]. It supports the definition and execution of processes on top of (web) services but also allows to implement continuously running processes for analyzing, processing, and managing data streams in real-time. Since processing data streams for evaluating the patient's health state requires the invocation of computationally intensive services, Grid concepts are exploited to support the distributed computation on top of heterogenous resources. Therefore, the different data streams coming from the various sensors of a patient are distributed within the Grid for parallel processing. Finally, the streams have to be joined in order to combine different sensor signals for rating medical relevance. The combination of process management and Grid concepts allows for the composition of existing services and for the efficient distribution of single service invocations within the Grid.

3 Data Stream Management for Medical Digital Libraries

In this section, we introduce an extended hyperdatabase system for the support and management of continuous data streams.

3.1 Challenges in Data Stream Management

The main challenges in *data stream management* (DSM) are imposed by the large number of sensors, components, devices, information systems and platforms connected by different network technologies, and by the vast amount of continuously generated data. For processing this data, existing systems and components are well in place and need to be incorporated into digital libraries. Reliability and provable correctness are new challenges that are of utmost importance particularly in healthcare applications, where failures may have perilous consequences. As described in Section 2, DSM has to interact with traditional process management in order to react to certain results (e.g., calling the ambulance) or to offer the user appropriate processes for the evaluation of DSM results. These challenges necessitate an infrastructure that combines the processing of data streams and process management, i.e., the possibility to combine services (conventional services as offered by digital libraries and services operating on data streams produced by sensors) and to execute composite services in a reliable way. Therefore, we propose an integrated information management infrastructure supporting user-defined processes, both conventional and processes performing DSM. *Hyperdatabase (HDB)* systems already provide an infrastructure for reliable process execution, which we will extend to enable DSM processes.

3.2 Peer-to-Peer Process Execution in the Hyperdatabase OSIRIS

A *hyperdatabase (HDB)* [6] is an infrastructure that supports the definition and reliable execution of user-defined processes on top of distributed components using existing services. Characteristic features of HDB's are the possibility to i.) add transactional guarantees to the execution of processes [7], ii.) support reliable peer-to-peer execution of processes without global control, thereby supporting a high degree of availability and scalability, and iii.) apply decentralized process execution in areas of intermitted connectivity.

OSIRIS (Open Service Infrastructure for Reliable and Integrated process Support) [4] is a prototype of a hyperdatabase, that has been developed at ETH Zurich and is used as a starting point of our joint HDB and Grid infrastructure. OSIRIS follows a novel architecture for distributed and decentralized process management. OSIRIS supports process execution in a peer-to-peer style based on locally replicated metadata, without contacting any central instance *(Peer-to-Peer Execution of Processes, P2PEP)*. With P2PEP, a component works off its part of a process and then directly migrates the instance data to nodes offering a suitable service for the next step(s) of the process according to its control flow specification. This is achieved by implementing two layers: the *HDB-layer*, a small software layer that is installed on each component providing a service

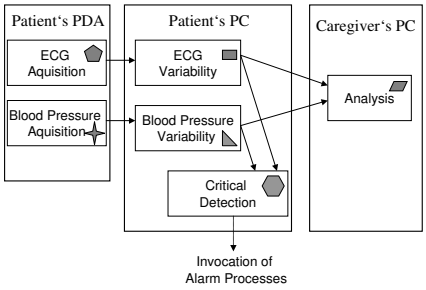


Fig. 2. Stream Process Processing ECG and Blood Pressure

and a set of global HDB repositories. These HDB repositories collect metadata on the processes to be executed, on the available components, and on their load. This meta information is smoothly distributed to the individual HDB layers – only metadata needed locally is actually replicated (e.g., only information on services and providers which might be invoked in some process are required at the local HDB layer of a component). More information on hyperdatabases and OSIRIS can be found in [6, 8, 4].

3.3 DSM enabled Extended Hyperdatabases Infrastructure

HDB's have to be extended in order to enrich their benefits with the capabilities for DSM [9]. We consider *stream-processes*, which perform continuous processing of data streams. The requirements for the execution of these stream processes are similar to those of conventional processes with respect to important aspects like distributed execution, load balancing, meta information distribution, or fault tolerance. Figure 2 illustrates a stream-process, which continuously processes patient's ECG and blood pressure. Sensor signals are recorded and preprocessed by patient's PDA, which is wirelessly connected to patient's PC. The PC does further processing and detects critical health conditions. Processed sensor information is continuously forwarded to the caregiver for further analysis.

Operators are the processing units of DSM. Operators perform stream operations on incoming data streams and produce outgoing data streams. Sensors are the primary sources of data streams. Sensors can be considered as operators without incoming data streams. DSM is done by combining operators, similar to the combination of activities in traditional process management. A stream-process is such a well defined set of logically linked operators continuously processing the selected input data streams, thereby producing results and having *side effects*. Side effects are effects on external systems imposed by processing results (e.g., feeding a digital library with medical relevant information gained by the stream process).

Based on the OSIRIS approach to fault-tolerant distributed peer-to-peer process execution, we need to distribute necessary meta information on stream processes for DSM in the same way this is done also for process management. This metadata contains the pieces of the global stream-process definition and a list of

offered stream operators of components, which are subject for smooth distribution among the suitable components offering the corresponding stream operators. A stream-process is set up by sending an activation message to the HDB-layer of the component hosting a source operator (e.g., the component is attached to a sensor or has a data stream input). Due to locally available metadata, the local HDB-Layer knows the subsequent stream operator and components, which offer these operators and is able to make the routing decision. Then the component sends an activation message to the selected subsequent components and provides them with needed data streams.

Our extended infrastructure also allows for load balancing during the execution of stream processes. Therefore, the distribution of metadata on the load of components that are able to host stream operators needs to be published. This load information is used to choose the best component during the stream-process activation. In case of high load, the overloaded component is able to transfer a running stream operator to a component with less load. When stream operations are affected that accumulate an internal state during their execution, this state has to be managed and transferred to the new host. Due to this fact, components make a backup of internal state of running stream-operators at a regular basis. Information about the backup location address is metadata, which is also smoothly distributed.

The previous techniques are also responsible to allow for sophisticated failure handling. In case a component hosting a stream operator fails, components hosting preceding parts of the same stream-process will recognize the failure because the transmission of their outgoing streams is no longer acknowledged. The infrastructure distinguishes between four failure cases. First, the failed component recovers within a certain timeout, then processing is continued in the state before the failure. This is possible since output queues of preceding components are used to buffer the data streams until they are acknowledged. Second, the failed component does not recover within the timeout period. In this case, the preceding component is in a similar situation as during the setup phase of the process. The component has to find suitable components that are able to perform subsequent stream operators. Due to local metadata, the new component is able to find the backup location and to load the old internal state for the continuation of stream processing. If the failed component recovers after the timeout, it has to be informed that its workload moved and that it is no longer in charge. Third, the failed component does not recover and there is no other suitable component. In this case, the stream-process may have an alternative processing branch (defined in the streaming process), which is now activated by the preceding component. Fourth, there is no recovery and no possibility to continue stream processing. If so, a conventional process can be invoked to handle the failure situation (e.g., calling an administrator to fix the problem).

This extended HDB system is capable of supporting telemonitoring applications by providing integrated process and data stream management in peer-to-peer style. Furthermore, it allows to seamlessly cooperate with digital libraries, e.g., by making use of the services that are provided to access information.

4 Digital Libraries on the Grid

An important challenge when dealing with service composition, especially with computationally complex services, is the efficient routing of service requests among a set of providers. OGSA (Open Grid Services Architecture) [10] compliant Grid systems are rapidly emerging and are widely accepted. These Grid systems provide support to efficiently invoke and use individual services in the Grid in a request/reply style. However, they do not support service composition and process execution. In contrast, the focus of state-of-the-art process support systems is not at all or only marginally oriented towards a tight integration into a Grid Environment.

4.1 Bringing Service Composition to the Grid

Although OSIRIS, the starting point of our integrated *DSM* and *Grid Infrastructure*, is quite powerful in doing distributed process management, it does not yet follow OGSA or WS-RF [11], the de facto standard for Grid Environments. It does also not make use of the enhanced features offered in the globus toolkit [5] (the reference implementation of OGSA) like, for example, resource management and security. In our current work, we aim to bring support for service composition to the Grid, which is done by extracting some of the ideas that can be found in OSIRIS, and integrate those with current standards and services which have recently emerged in the Grid Community. This will result in a set of new OGSA compliant services enhancing current Grid Infrastructures with the ability of recursive process composition.

There are several possibilities to decompose an application into smaller parts that can then be executed in parallel. The most important ones are master/slave type of applications, as well as the divide and conquer or branch and bound paradigms. The applicability of these paradigms of course strongly depends on the semantics of the application to be parallelized. Especially the master/slave paradigm is very suitable to Grid-enable applications [12], and is therefore widely used. In case of master/slave parallelization, the main prerequisites are: i) few or *no communication* among the sub parts ii) work is dividable among *identical* sub parts iii) work can be dis- and reassembled in a central point iv) work can be parameterized and parallelized and does not need serial iterative processing.

Since the potential for master/slave parallelization can be found in several applications, we have started to apply this paradigm to enhance the efficiency, the creation, and the ease-of-use of services in the Grid. Using the master/slave paradigm, applications developers can focus on the implementation of the problem specific subparts of the service as well as on the split into and merge of parallel subparts, but they do *not* need to care about the distribuiton of subparts. This is particularly important since the latter requires dynamic information on the currently available resources which is not available at build-time, when the services, their split and merge are defined.

4.2 The Frameworks Architecture and Use

To ease the creation of services for tomorrow's Grid Infrastructures, we are currently developing a generic framework to handle master/slave applications where a single master process controls the distribution of work to a set of identically operating slave processes. This framework is designed to accept ordinary Web/Grid Services as destinations for calls, as well as composite services. The framework enables application developers to port new master/slave type of applications to the Grid by just implementing a very limited set of *application focused methods*, and declare the so implemented classes as available to the framework in a deployment descriptor file. The framework takes care about all the *infrastructure related functionality* like marshaling and unmarshaling, communications, failure handling and distributed invocation of services depending on availability and performance considerations. Among this functionality, the framework dynamically takes care about the level of parallelization based on the current status of the Grid, availability of nodes, and QoS restrictions.

The framework developed is based on GT3 [5]. The core part consists of a set of classes building the central master and slave services. These are OGSA-compliant Grid Services [10] bundled with corresponding stubs and some supporting classes for specialized exceptions and encapsulating the input and output parameters passed around. The work left to the application programmer is to implement abstract methods which are responsible for the application specific part, in particular methods for splitting, merging, and the actual application logic. The ones for splitting and merging used in the master service, and the calculative method is used to concretely specify what the slave has to do. In addition, a Web Service deployment descriptor (WSDD) has to be written, as specified by the Axis framework [13], which GT3 is partly based on. At runtime, the framework determines which slaves to use, out of the set of all slaves registered to provide the appropriate service. This is done by accessing an *IndexService* available in the Grid. The request is then forwarded to all the slaves, after being divided into subtasks. This is shown in the upper right corner of figure 3 where the service depicted as cross is provided by a set of slave services executing in parallel.

The current implementation can easily be adopted to more sophisticated distribution mechanisms based on the Service Data Elements (SDE's) [5] provided by each Grid Service. There might be more specialized implementations that distribute to slaves based on current workload, cost or other metrics available. After having distributed he work, the *MasterService* registers for notifications from the slaves and waits for results. After all slaves have returned, the Master Service generates the final result by merging the results of the subparts and returns the completed result to the requestor. An important aspect here is to provide sophisticated failure handling that allows the Master Service to re-distribute requests when slaves have failed during the execution of their subpart. On the slaves side, in addition to the implementation of the actual application logic, a deployment descriptor is needed that specifies where to register this particular slave service.

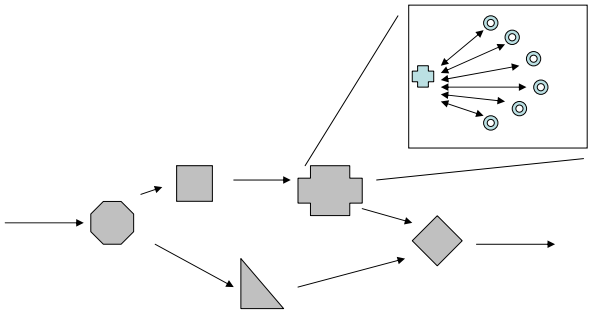


Fig. 3. Process containing a dynamically acting Grid Node

In the scenario described in section 2, there is one master Grid Service which accepts streamed data from the patients life vest and ECG. This service acts, from the point of the process management system, as an ordinary step in the process chain. However, in the background it re-directs the data stream to the slaves available in the system and checks the data against local replicas of digital libraries holding characteristic pathologic and non pathologic data. The time intensive comparison of the streamed data with entries in the digital library is done in a distributed way on the Grid. The slaves report the result of their search back to the master who is then able to store the data for further usage and to trigger subsequent services or processes when needed (e.g., in critical situations).

4.3 From Master/Slave to Process Execution

A *MasterService* can generally be seen as a Grid Service that controls the execution and dataflow among a set of several services whose availability, number and distribution is only known during runtime and subject to frequent changes. Since from the point of view of the OSIRIS process execution engine, it acts just as any other operator or service, the dynamics of request distribution as well as the distribution pattern itself is transparent to the process execution engine. Figure 3 illustrates a process schema as executed by OSIRIS including a dynamically acting Grid Node. One step in this process, shown as a cross, is dispatching the request to various nodes in the Grid and awaits their feedback. The process execution engine is not aware of this dispatching behind the scenes. This leads to the more general idea that the *MasterService* can be seen as a Process Execution Service itself, calling arbitrary Grid Services — either in parallel, sequentially or in any other pattern available to the system.

This Process Execution Services can be deployed to the Grid as highly dynamic components. The distribution pattern of an algorithm can be determined at runtime based on some QoS information provided through the caller or can be hard wired to a special distribution pattern.

In order to avoid a centralized Process Execution Service that could lead to a single point of failure, we are currently integrating the distributed process exe-

cution engine described in OSIRIS. In OSIRIS, the execution plan for a process (determined by the control flow) is, prior to its invocation, split up into several execution steps. Each step consists of a service invocation, and information of all successors. This allows to move the control from a centralized component to the responsibility of each node participating in the process. Therefore, this approach is much more robust to the failure of single nodes execution and triggering the next step is up and running) than centralized solutions.

5 Related Work

5.1 Data Stream Management

DSM aspects are addressed by various projects like NiagaraCQ [14], STREAM [15], and COUGAR [16]. The main focus of these projects is on query optimization and approximate query results and data provided by sensor networks. Aurora [17] allows for user defined query processing by placing and connecting operators in a query plan. Aurora is a single node architecture, where a centralized scheduler determines which operator to run. Extensions like Aurora* and *Medusa* [18] also address DSM in distributed environments. TelegraphCQ [19] is a DSM project with special focus on adaptive query processing. Fjords allow for inter-module communication between an extensible set of operators enabling static and streaming data sources. Flux [20] provides load balancing and fault tolerance. PeerCQ [21] is a system that offers a decentralized peer-to-peer approach supporting continual queries running in a network of peers. The DFuse [22] framework supports distributed data fusion. Compared to other projects in this field, our infrastructure offers two unique characteristics. Firstly, dynamic peer-to-peer process execution where local execution is possible without centralized control. Secondly, the combination of DSM and transactional process management enables sophisticated failure handling.

5.2 Grid Infrastructure

The master/slave paradigm is commonly agreed as valuable asset for the development of Grid applications [12]. The master-worker tool [23] provides the possibility to integrate applications in the Grid by implementing a small number of user-defined functions concentrating on the applications main purpose. It is applied to complex problems from the field of numerical optimization [24]. While it is tightly integrated into a former Grid environment, the Globus Toolkit 2, our approach uses more recently emerged technologies and focuses on evolving into a more generally useable distributed process execution engine.

A similar approach is taken in AppLeS Master-Worker Application Template (AMWAT) [25] where the main emphasis is on scheduling issues and a workflow model to select the best locations for the master and worker services. Other Approaches focusing on other task-parallel models can be found in [26, 27] for the divide-and-conquer distribution pattern, and [28] for branch-and-bound.

In [29], BPEL4WS, the Business Process Execution Language for Web Services [30] is evaluated for the use within transactional business processes on the Grid. The authors point out that the usage of single, non-orchestrated Web Services is limited, and that there is a need for reliable and coordinated process execution on the Grid.

6 Conclusion and Outlook

The proliferation of ubiquitous computing and the huge amount of existing information sources is leading towards a world where sophisticated information management is becoming a crucial requirement. A digital library for medical applications not only has to manage discrete data, it has also to support the acquisition, processing, and storage of streaming information that is continuously produced by sensors. Essentially, both streaming and non-streaming processes and applications have to be supported. Moreover, due to the complex processing operators that are used within stream processes, the distribution of work is a major requirement to efficiently process continuous data streams. By exploiting the features of a Grid infrastructure, subparts can be executed in parallel by making use of the resources that are available at run-time. As a paradigm for the distribution of work within the Grid, we have integrated a master/slave type of interaction into a stream-enabled HDB system.

Based on this extended HDB system, we are currently building a comprehensive infrastructure that jointly addresses process-based service composition and streaming processes, and that is enriched by features from an existing Grid infrastructure. In terms of the distribution paradigms supported, we are currently extending the master/slave type of distribution to allow for arbitrary execution plans. The goal is to define a generic, distributed and OGSA compliant process execution engine. This engine has to support different control flow specifications for composite services that are controlled by the Grid-enabled Process Execution Services so that it can be exploited for process-based applications on top of medical digital libraries.

References

1. Haux, R., Kulikovski, C.: Digital Libraries and Medicine. Yearbook of Medical Informatics (2001)
2. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1995)
3. VivoMetrics: VivoMetrics – Continuous Ambulatory Monitoring. http://www.vivometrics.com/site/system.html (2003)
4. Schuler, C., et al.: Peer-to-Peer Process Execution with OSIRIS. In: Proceedings of ICSOC 2003, Trento, Italy, Springer LNCS, Vol. 2910 (2003) 483–498
5. The Globus Alliance: The Globus Toolkit Version 3. http://www-unix.globus.org/toolkit/ (2003)
6. Schek, H.J., Böhm, K., Grabs, T., Röhm, U., Schuldt, H., Weber, R.: Hyperdatabases. In: Proc. of WISE Conf., Hong Kong, China (2000) 28–40

7. Schuldt, H., Alonso, G., Beeri, C., Schek, H.J.: Atomicity and Isolation for Transactional Processes. ACM Transactions on Database Systems **27** (2002) 63–116
8. Schek, H.J., Schuldt, H., Schuler, C., Weber, R.: Infrastructure for Information Spaces. In: Proc. of ADBIS Conf., Bratislava, Slovakia (2002) 22–36
9. Brettlecker, G., Schuldt, H., Schatz, R.: Hyperdatabases for Peer–to–Peer Data Stream Management. In: Proc. of ICWS Conf., San Diego, USA (2004) to appear
10. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration (2002)
11. Sabbah, D.: Bringing Grid & Web Services Together. Presentation, IBM Software Group (2004) http://www.globus.org/wsrf/sabbah_wsrf.pdf.
12. Foster, I., Kesselman, C., eds.: The Grid 2, Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers (2004)
13. Apache WebServices Project: AXIS. http://ws.apache.org/axis/ (2004)
14. Chen, J., et al.: NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In: Proc. of SIGMOD Conf., Dallas, TX, USA (2000) 379–390
15. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and Issues in Data Stream Systems. In: Proc. of PODS Conf., Madison, WI, USA (2002) 1–16
16. Yao, Y., Gehrke, J.: Query Processing for Sensor Networks. In: Proc. of CIDR Conf., Asilomar, CA, USA (2003)
17. Carney, D., et al.: Monitoring Streams - A New Class of Data Management Applications. In: Proc. of VLDB Conf., Hong Kong, China (2002) 215–226
18. Cherniack, M., et al.: Scalable Distributed Stream Processing. In: Proc. of CIDR Conf., Asilomar, CA, USA (2003)
19. Chandrasekaran, S., et al.: TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In: Proc. of CIDR Conf., Asilomar, CA, USA (2003)
20. Shah, M., et al.: Flux: An Adaptive Partitioning Operator for Continuous Query Systems. In: Proc. of ICDE Conf., Bangalore, India (2003)
21. Gedik, B., et al.: PeerCQ:A Decentralized and Self-Configuring Peer-to-Peer Information Monitoring System. In: Proc. of Distributed Computing Systems Conf., Providence, RI, USA (2003) 490–499
22. Kumar, R., et al.: DFuse: a Framework for Distributed Data Fusion. In: Proc. of SensSys Conf., Los Angeles, CA, USA (2003) 114–125
23. Linderoth, J., et al.: An Enabling Framework for Master - Worker Applications on the Computational Grid. In: 9th IEEE Int'l Symp. on High Performance Dist. Comp., Los Alamitos, CA, IEE Computer Society Press (2000) 43–50
24. Anstreicher, K., et al.: Solving Large Quadratic Assignment Problems on Computational Grids. In: Mathematical Programming 91(3). (2002) 563–588
25. Shao, G.: Adaptive Scheduling of Master/Worker Applications on Distributed Computational Resources. PhD thesis, University of California - San Diego (2001)
26. Foster, I.: Automatic Generation of Self - Scheduling Programs. In: IEEE Transactions on Parallel and Distributed Systems 2(1). (1991) 68–78
27. v. Nieuwpoort, R., et al.: Efficient Load Balancing for Wide - Area Divide - And - Conquer Applications. In: 8th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. (2001) 34–43
28. Iamnitchi, A., et al.: A Problem-specific Fault-tolerance Mechanism for Asynchronous Distributed Systems. In: Int'l Conference on Parallel Processing. (2000)
29. Leymann, F., Güntzel, K.: The Business Grid: Providing Transactional Business Processes via Grid Services. In: Proc. of ICSOC 2003, (Springer)
30. Andrews, T., et al.: Business Process Execution Language for Web Services (BPEL4WS) 1.1. BEA, IBM, Microsoft, SP, Siebel. (2003)

Supporting Information Access in Next Generation Digital Library Architectures*

Ingo Frommholz, Predrag Knežević, Bhaskar Mehta, Claudia Niederée, Thomas Risse, and Ulrich Thiel

Fraunhofer IPSI
Integrated Publication and Information Systems Institute
Dolivostrasse 15, 64293 Darmstadt, Germany
{frommholz|knezevic|mehta|niederee|risse|thiel}@ipsi.fhg.de

Abstract. Current developments on Service-oriented Architectures, Peer-to-Peer and Grid computing promise more open and flexible architectures for digital libraries. They will open DL technology to a wider clientele, allow faster adaptability and enable the usage of federative models on content and service provision. These technologies rise new challenges for the realization of DL functionalities, which are rooted in the increased heterogeneity of content, services and metadata, in the higher degree of distribution and dynamics, as well as in the omission of a central control instance. This paper discusses these opportunities and challenges for three central types of DL functionality revolving around information access: metadata management, retrieval functionality, and personalization services.

1 Introduction

Currently, there is a considerable amount of R&D activity in developing viable strategies to use innovative technologies and paradigms like Peer-to-Peer Networking, Grid, and Service-oriented Architectures in digital libraries (see e.g. the European Integrated Projects BRICKS [1] and DILIGENT). The promise is that these efforts will lead to more open and flexible digital library architectures that:

- open up digital library (DL) technology to a wider clientele by enabling more cost-effective and better tailored digital libraries,
- allow faster adaptability to developments in DL services and IT technologies, and
- enable usage of dynamic federative models of content and service provision involving a wide range of distributed content and service providers.

The use of Service-oriented Architectures, Grid infrastructures, and the Peer-to-Peer approach for content and service provision has implications for the realization of enhanced DL functionality. These implications are mainly rooted in increased heterogeneity of content, services and metadata, in the higher degree of distribution and dynamics, as well as in the omission of a central control instance. On one hand, these are opportunities for better and more multifarious DL services; on the other hand, these are

* This work is partly funded by the European Commission under BRICKS (IST 507457), COLLATE (IST-1999-20882), DILIGENT and VIKEF (IST-507173)

new challenges to ensuring long-term, reliable, and quality-ensured DL service provision that also exploits the technology promises. This paper discusses these opportunities and challenges for three central types of DL functionality revolving around information access: metadata management, retrieval functionality, and personalization services.

The rest of this paper is structured as follows: Section 2 presents the key ideas of next generation DL architectures based on exemplary RTD projects. Section 3 discusses how these new ideas influence information access in the areas of metadata management, information retrieval, and personalization support. Related work in these areas is considered in section 4. The paper concludes with a summary of the paper's key issues.

2 Next Generation Digital Library Architectures

Current plans for next generation DL architectures are aiming for a transition from the DL as an integrated, centrally controlled system to a dynamic configurable federation of DL services and information collections. This transition is inspired by new technology trends and developments. This includes technologies like Web services and the Grid as well as the success of new paradigms like Peer-to-Peer Networking and Service-oriented Architectures. The transition is also driven by the needs of the "DL market":

- better and adaptive tailoring of the content and service offer of a DL to the needs of the respective community as well as to the current service and content offer;
- more systematic exploitation of existing resources like information collections, metadata collections, services, and computational resources;
- opening up of DL technology to a wider clientele by enabling more cost-effective digital libraries.

To make these ideas more tangible we discuss three RTD projects in the field and discuss the relationship to upcoming e-Science activties.

2.1 Virtual Digital Libraries in a Grid-based DL Infrastructure

DILIGENT[1] is an Integrated Project within the IST 6th Framework Programme. It's objective is "to create an advanced test-bed that will allow members of dynamic virtual e-Science organizations to access shared knowledge and to collaborate in a secure, coordinated, dynamic and cost-effective way."

The DILIGENT testbed will enable the dynamic creation and management of Virtual Digital Libraries (VDLs) on top of a shared Grid-enabled DL infrastructure, the DILIGENT infrastructure. VDLs are DLs tailored to the support of specific e-Science communities and work groups. For creating a VDL, DL services, content collections, metadata collections are considered as Grid resources and are selected, configured, and integrated into processes using the services of the DILIGENT infrastructure. This infrastructure builds upon an advanced underlying Grid infrastructure as it is currently evolving e.g. in the EGEE project[2]. Such a Grid infrastructure will already provide parts of the functionality required for DILIGENT. This includes the dynamic allocation

[1] DILIGENT - A DIgital Library Infrastructure on Grid ENabled Technology
[2] http://public.eu-egee.org

of resources, support for cross-organizational resource sharing, and a basic security infrastructure. For effectively supporting DLs, additional services like support for redundant storage and automatic data distribution, metadata broker, metadata and content management, advanced resource brokers, approaches for ensuring content security in distributed environments and the management of content and community workflows are rquired in addition to services that support the creation and management of VDLs. A further project challenge are systematic method to make the treasure of existing DL services and collections utilizable as Grid resources in the DILIGENT infrastructure.

The DILIGENT project will result in a Grid-enabled DL testbed that will be validated by two complemtary real-life application scenarios: one from the Cultural Heritage domain and one from the environmental e-Science domain.

2.2 Service-oriented and Decentralized DL Infrastructure

The aim of the BRICKS[3] Integrated Project [1] is to design, develop and maintain a user and service-oriented space to share knowledge and resources in the Cultural Heritage domain. The target audience is very broad and heterogeneous and involves cultural heritage and educational institutions, research community, industry, and citizens.

Such high level of heterogeneity cannot be handled with the existing centralized DL architectures. The BRICKS architecture will reduce the cost to join the system, i.e. the system will reuse existing communication channels and content of already installed DLs. Also, the BRICKS membership will be flexible, such that parties can join or leave the system at any point in time without administrative overheads. The BRICKS project will define a decentralized, service-oriented infrastructure that uses Internet as a backbone and fulfills the requirements of expandability, scalability and interoperability.

With respect to access functionality, BRICKS provides appropriate task-based functionality for indexing/annotation and collaborative activities e.g. for preparing a joint multimedia publication. An automatic annotation service will enable users to request background information, even if items have not been annotated by other users yet. By selecting appropriate items, such as definitions of concepts, survey articles or maps of relevant geographical areas, the service exploits the currently focussed items and the user's goals expressed in the user profile. In addition, the linking information, which is generated dynamically, must be integrated into the documents. The design of the access functionality is influenced by our experiences in the 5th Framework project COLLATE.

2.3 COLLATE: A Web-based environment for document-centered collaboration

Designed as a content- and context-based knowledge working environment for distributed user groups, the COLLATE system supports both individual work and collaboration of domain experts with material in the data repository. The example application focuses on historic film documentation, but the developed tools are designed to be generic and as such adaptable to other content domains and application types. This is achieved by model-based modules.

[3] BRICKS - Building Resources for Integrated Cultural Knowledge Services

The system supports collaborative activities such as creating a joint publication or assembling and creating material for a (virtual) exhibition, contributing unpublished parts of work in the form of extended annotations and commentaries. Automatic indexing of textual and pictorial parts of a document can be invoked. Automatic layout analysis for scanned documents can be used to link an annotation of individual segments. As a multifunctional means of in-depth analysis, annotations can be made individually but also collaboratively, for example in the form of annotation of annotations, collaborative evaluation, and comparison of documents. Through interrelated annotations users can enter into a discourse on the interpretation of documents and document passages.

The COLLATE collaboratory is a multifunctional software package integrating a large variety of functionalities that are provided by cooperating software modules residing on different servers. It can be regarded as a prototypical implementation of a decentralized, Service-oriented DL architecture which serves as a testbed for the collaborative use of documents and collections in the Humanities. The collaborative creation of annotation contexts for documents offers new opportunities for improving the access functionality, as we will illustrate later on.

2.4 Next Generation DL Architectures and e-Science

Scientific practice is increasingly reliant on data-intensive research and international collaboration enabled by computer networks. The technology deployed in such scenarios allows for high bandwidth communication networks, and by linking computers in ”Grids” places considerably more powerful computing resources is at their disposal than a single institution could afford. If we view e-Science as being primarily motivated up to now by notions of resource sharing for computationally intensive processes (e.g. simulations, visualisation, data mining) a need is emerging for new approaches, brought up by ever more complex procedures, which, on the one hand, assume the reuse of workflows, data and information and, on the other hand, should be able to support collaboration in virtual teams. Future concepts of e-Science will be less focussed on data and computing resources, but will include services on the knowledge and organizational levels as well. Embedding future DL architectures in an emerging e-Science infrastructure will meet these requirements by providing access to information and knowledge sources, and appropriate collaboration support on top of the Grid-based infrastructure.

3 Information Access in Next Generation DL Architectures

A decentralized, service-oriented architecture poses new challenges to the technologies employed for information access. DLs based on such an architecture should, for example, not only provide access and retrieval functionality for the documents residing on the local peer, but should also consider other peers which might host relevant document w.r.t. a query. In the following, we will outline possible approaches for enhanced services for information access. Such services will utilize the functions of a decentralized metadata management ensuring the availability of all documents (and their parts) while reducing overhead costs. Retrieval functions can be improved by taking into account

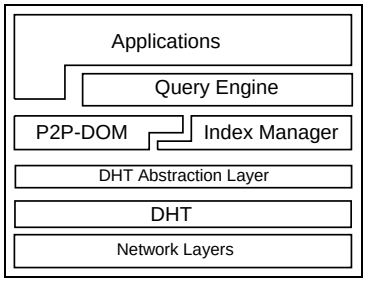


Fig. 1. Decentralized XML Storage Architecture

the annotational contexts of documents emerging for the collaborative process of interpreting and discussing items of interests by a group of users. In addition, individual users‘ contexts can be used to personalize the access services.

3.1 Decentralized Metadata Management

DLs usually like to keep content under control in their local repositories. On the contrary, metadata should be available for all parties, stored in some central place accessible for everybody. Decentralized architectures by definitions avoid having central points, for the following reasons: they are candidate single point of failure and performance bottleneck. Therefore, metadata must be spread in the community. A naïve approach for metadata searching would be to distribute queries to all members, but it is obvious that the solution is unscalable. Hence, efficient metadata access and querying are very important challenges withing the new decentralized settings.

Our proposal to these challenges is a decentralized Peer-to-Peer datastore that will be used for managing XML-encoded metadata. It balances resource usage within the community, has high data availability (i.e. data are accessible even if creator disappears from the system, e.g. system fault, network partitioning, or going offline), is updateable (i.e. stored data can be modified during the system lifetime), and supports a powerful query language (e.g XPath/XQuery).

XML documents are split into finer pieces that are spread within the community. The documents are created and modified by the community members, and can be accessed from any peer in a uniform way, e.g. a peer does not have to know anything about the data allocation. Uniform access and balanced storage usage are achieved by using a DHT (Distributed Hash Table) Overlay [2] and having unique IDs for different document parts.

Figure 1 shows the proposed storage architecture, where all layers exist on every peer. The datastore is accessed through the P2P-DOM component or by using the query engine that could be supported by an optional index manager. A more detailed discussion about the proposed approach, challenges and open issues can be found in [3].

In the rest of the subsection, we are giving more details how the proposed datastore could be used for managing service metadata, which are an additional type of DL metadata introduced by Service-oriented Architectures.

Service metadata describe service functionalities, interfaces and other properties. These meta-information are usually encoded by using WSDL (Web Service Description Language [4]) and published to an UDDI (Universal Description, Discovery and Integration [5]) service directory. Service discovery queries are usually more complex than simple name matching, i.e. they contain qualified, range and/or boolean predicates.

In order to realize a decentralized service directory with advanced query mechanisms, the community of service providers will create and maintain in the decentralized P2P data store a pool of the service descriptions. Every service will be able to modify its description during the lifetime and to search for needed services. Query execution will be spread at many peers, the query originator will only get the final result back.

At the same time, due to uniform data access, new community members can start using the service directory immediately after joining the system without additional setup and administration. A member decision to leave the community will not make any influence for the rest of the system, because data are replicated. Even if network partitioning happens, the service directory would provide access to service metadata available in the partition allowing some parties to continue with work without interruption.

For details about the use of the decentralized datastore in other scenarios see [6].

3.2 Decentralized Context-based Information Retrieval

DLs based on a decentralised architecture should not only provide access and retrieval functionality for the documents residing on the local peer, but should also consider other peers which might host relevant document w.r.t. a query. It is clear that for a scenario like described above appropriate search functionality has to be defined. In the following, we will outline possible approaches for enhanced retrieval services.

Services In order to be able to abstract from the underlying infrastructure, retrieval functionality should be implemented as a service with a predefined API and behaviour. This has the advantage that other peers are able to query the local repository, which is an important feature for enabling P2PIR. An example Web Service specification for search and retrieval is SRW[4]. It considers content-based retrieval functionality, but lacks context-based features as proposed above. When performing retrieval based on the annotation context (see below), such context information should be contained in the result set in order to elucidate why an item was retrieved. So a common API for queries, results and indexing requests has to be identified which is capable of taking advanced queries and context information into account.

Annotation Context Annotations are a certain kind of metadata providing some information about the annotated document. They can contain content about content (e.g., interpretations, comments), other information like judgements, or references to other documents [7]. Annotations can be either manually or automatically created.

Manual annotations range from personal to shared to public ones. They can include personal notes, e.g., for comprehension, and whole discussions about documents [8, 9]. Annotations are building blocks for collaboration. In a distributed, decentralized environment, especially shared and public annotations pose a challenge to the underlying

[4] `http://www.loc.gov/z3950/agency/zing/srw/`

services. Users can create shared and public annotations residing on their peers, but this data has to be spread to other peers as well.

By automatic annotations, we mean the automatic creation and maintenance of annotations consisting of links to and summaries of documents on other peers which are similar to documents residing on the local peer. Such annotations constitute a context in which documents on a peer are embedded. For each document, agents could be triggered to periodically update the information at hand, similar to the internal linking methods like similarity search, enrichment and query generation proposed in [10]. P2PIR methods can possibly be applied for this. The underlying assumption is that a user stores potential interesting documents on her peer and is interested in similar publications. Automatic annotations can be created w.r.t. several aspects. For instance, topical similar documents can be sought after. Another interesting kind of automatic annotation can be extracted from the surroundings of a citation. If a document residing on another peer cites a document on the local peer, the surroundings of this citation usually contain some comments about the cited document (similar as reported in [11]). Since only annotations to documents residing on the peer are created, storage costs can be kept low. Regular updates performed by agents keep the user informed.

Annotations, either manual or automatic ones, constitute a certain kind of *document context*. Annotation-based retrieval methods [8] can employ the annotation context without the need to actual access other peers. Since annotations, being manually or automatically created, contain additional information about the document, we assert that annotation-based retrieval functions boost retrieval effectiveness. Future work will show if this assumption holds. Using annotations for information retrieval in a decentralized environment has the advantage that annotations are locally available, but reflect information lying on other peers. In this way, annotations create new access structures which help adressing problems arising when performing information retrieval on an underlying P2P infrastructure.

3.3 Cross-Service Personalization

Personalization approaches in DLs dynamically adapt the community-oriented service and content offerings of a DL to the preferences and requirements of individuals [12]. They enable more targeted information access by collecting information about users and by using these user models (also called user profiles) in information mediation.

Personalization typically comes as an integral part of a larger system. User profiles are collected based on a good knowledge about the meaning of user behavior and personalization activities are tailored to the functionality of the respective system. Within a next-generation distributed DL environment, which is rather a dynamic federation of library services than a uniform system, there are at least two ways to introduce personalization. In the simple case, each service component separately takes care of its personalization independently collecting information about users. A more fruitful approach, however, is to achieve personalization across the boundaries of individual services, i.e., cross-system or, more precisely, cross-service personalization. In this case, personalization relies on a more comprehensive picture of the user collected from his interaction with different library services.

Cross-service Personalization Challenges Cross-service personalization raises the following challenges: How to bring together the information about a user and his interactions collected by the different services in a comprehensive way and make up-to-date information about the user available? How to manage, update, and disseminate user models to make them accessible to the different services? How to support (at least partial) interpretation of the user model in a heterogeneous, and dynamically changing DL service environment? This requires a shared underlying understanding of the user model. Furthermore, it raises issues of privacy and security, since personal data is moved around in a distributed system.

Approaches to Cross-Service Personalization We identified two principle approaches which differ from each other in their architecture. A flexible and extensible user model that can capture various characteristics of the user and his/her context is in the core of both approaches. We call the operationalization of such a model *context passport* [13] in what follows, implying that it is accompanies the user and is "presented" to services to enable personalized support. The idea of the context passport is discussed in more detail after presenting the two approaches:

Adaptor approach: The adaptor approach relies on the ideas of wrapper architectures. A kind of wrapper is used to translate information access operations into personalized operations based on the information collected in the context passport. The advantage of this approach is that personalization can also be applied to services that themselves do not support personalization. The disadvantage is that every service will need its own wrapper. Unless there is a high degree of standardization in service interfaces, creating wrappers for every individual services may not be practical and does not scale well in dynamic service environments.

Connector approach: In contrast to the adaptor approach, the connector approach relies on the personalization capabilities of the individual services. It enables the bidirectional exchange of data collected about the user between the context passport and the personalization component of the respective service. The context passport is synchronized with individual user models/profiles maintained by services. The advantage here is that personalization of one service can benefit from the personalization efforts of another.

The context passport [13] is positioned as a temporal memory for information about the user. It covers an extensible set of facets modeling different user model dimensions, including cognitive pattern, task, relationship, and environment dimension. The context passport acts as an aggregated service-independent user profile with services receiving personalization data from the context passport. Services also report to the context passport based on relevant user interaction which add up-to-date information to the user's context. The context passport is maintained by an active user agent which communicates with the services via a specific protocol.

A flexible protocol is required for this communication between context passport and the service-specific personalization component. Such a protocol has to support the negotiation of the user model information to be exchanged and the bidirectional exchange of user information. As the services require different meta data about a user, there has to be a negotiation and an agreement between the service and the context passport about

what information is required. In order to keep the context passport up-to-date, the services needs to inform the context passport about tnew knowledge gained about the user. There is thus a requirement from bidirectional information exchange so that other services may benefit from up-to-date information about the user.

4 Related Work

Metadata Management Decentralized and peer-to-peer systems can be considered as a further generalization of distributed systems. Therefore, decentralized data management has much in common with distributed databases, which are already well explored [14, 15]. However, some important differences exist. Distributed databases are made to work in stable, well connected environments (e.g. LANs) with the global system overview, where every crashed node is eventually replaced by a new proper one. Also, they need some sort of administration and maintenance.

On the contrary, the P2P systems are deployed mostly on the highly unreliable Internet. Some links can be down, network bandwidths are not guaranteed. The P2P systems allow disconnection of any peer at any time, without a need for replacement, and none of the peers is aware of the complete system architecture. Therefore, the system must self-organize in order to survive such situations.

Many distributed databases like Teradata, Tandems NonStopSQL, Informix Online Xps, Oracle Parallel Server and IBM DB2 Parallel Edition [16] are available on the market. The first successful distributed filesystem was Network File System (NFS) succeeded by Andrew File System (AFS), Coda and xFS, etc.

Current popular P2P file-sharing systems (e.g. KaZaA, Gnutella, eDonkey, Past [2]) might be a good starting point for enabling decentralized data management. However, these systems have some important drawbacks: file-level granularity and write-once access, i.e. files are non-updateable after storing. Storing a new version requires a new filename. Usually, a file contains many objects. As a consequence, retrieving a specific object would require getting the whole file first. If a object must be updated, then a whole new file version must be created and stored. In current systems it is not possible to search for a particular object inside the files. The query results contain the whole files, not only requested objects. Advanced searching mechanism like qualified, range or boolean predicates search is not supported. Usually, metadata have rich and complex structure and queries on them are more than simple keyword match. Also, metadata should be updateable. Thus, the presented P2P systems are not suitable for decentralized metadata management.

There are some attempts [17] to extend Gnutella protocols to support other types of queries. It would be quite possible to create a Gnutella implementation that understands some variant of SQL, XPath or XQuery. However, such networks would have problems with system load, scalability and data consistency, e.g. only locally stored data could be updated and mechanisms for updating other replicas do not exist.

Information Retrieval Typical Peer-to-peer information retrieval (P2PIR) methods are working decentralized, as proposed by the P2P paradigm [2]. No server is involved as it would be in a hybrid or client-server architecture. Common P2PIR approaches let the

requesting peer contact other peers in the network for the desired documents. In the worst case, the query is broadcast to the whole network resulting in lots of communication overhead. Another approach would be to store all index information on every peer and search for relevant documents locally. Peers would request the required information during the inital introduction phase, and updates would be spread from time to time. However, this approach is not feasible since the expected storage costs would be quite high. Intermediate approaches which try to balance communication and storage costs work with peer content representations like the clustering approach discussed in [18]. Such a peer content representation does not need the amount of data a snapshot of the whole distributed index would need, but conveys enough information to estimate the probability that a documents relevant to the query can be found on a certain peer.

Some annotation systems [19] provide simple full-text search mechanisms on annotations. The Yawas system [20] offers some means to use annotations for document search, e.g. by enabling users to search for a specific document type considering annotations. Golovchinsky *et al.* [21] use annotations as markings given by users who judge certain parts of a document as being important when emphasizing them. Their approach gained better results than classic relevance feedback, as experiments showed. Agosti *et al.* [7] discuss facets of annotations and propose an annotation-based retrieval function based on probabilistic inference. The idea of automatic annotations is motivated by the internal linking methods described in [10] by Thiel *et al.*

Personalization Support The most popular personalization approaches in digital libraries or more general in information and content management systems are recommender systems and methods that can be summarized under the term personalized information access. Recommender systems (see e.g. [22]) give individual recommendations for information objects following an information push approach, whereas personalized information access (personalized newspapers, etc.) is realized as part of the information pull process, e.g. by filtering retrieval results or refining the queries themselves.

Personalization methods are based on modeling user characteristics, mainly cognitive pattern like user interests, skills and preferences [23]. More advanced user models also take into account user tasks [24] based on the assumption that the goals of users influence their needs. Such extended models are also referred to as user context models [25]. A flexible user context model that is able to capture an extensible set of user model facets as it is required for cross-service personalization can be found in [13]. Information for the user models (also called user profiles) are collected explicitly or implicitly [26], typically by tracking user behavior. These user profiles are used for personalized filtering in information dissemination (push) as well as in information access (pull) services. An important application area is personalized information retrieval. The information about the user is used for query rewriting [27], for the filtering of query results [28] as well as for a personalized ranking of query results [29].

5 Conclusions and Future Work

In this paper, we discussed opportunities and challenges for information access support resulting from the transition of more traditional, centrally controlled DL architectures

to DLs as dynamic federations of content collections and DL services. The discussion focussed on metadata management, information retrieval, and personalization support. In addition to discussing the central challenges, an advanced approach has been discussed for each of the three aspects: For metadata management a decentralized P2P data store solves the problem of systematic and efficient decentralized metadata management. Applications of annotations and annotation-based retrieval in the P2P context is considerd as a way to improved information retrival support in a decentralized environment. Finally, cross-service personalization is discussed as an adequate way to handle personalization in a dynamic service-oriented environment.

The list of the considered information access issues discussed is not meant to be exhaustive. Further challenges raise within next-generation DL architectures like effective metadata brokering and advanced methods for ensuring content security and quality. The envisaged support for information access needs to combine the approaches mentioned above in a balanced way to ensure that users will benefit from decentralized architectures, while at the same time, maintaining the high level of organization and reachability that users of DL systems are used to. Such issues are addressed in the BRICKS and the DIIGENT project in which our institute is involved together with partners from other European countries.

References

1. BRICKS Consortium: BRICKS - Building Resources for Integrated Cultural Knowledge Services (IST 507457). (2004) http://www.brickscommunity.org/.
2. Milojičić, D., Kalogeraki, V., Lukose, R., Nagaraja, K., Pruyne, J., Richard, B., Rollins, S., Xu, Z.: Peer-to-peer computing. Technical report (2002) http://www.hpl.hp.com/techreports/2002/HPL-2002-57.pdf.
3. Knežević, P.: Towards a reliable peer-to-peer xml database. In Lindner, W., Perego, A., eds.: Proceedings ICDE/EDBT Joint PhD Workshop 2004, P.O. Box 1527, 71110 Heraklion, Crete, Greece, Crete University Press (2004) 41–50
4. W3C: Web Services Description Language (WSDL) 1.1. (2001) http://www.w3.org/TR/wsdl.
5. OASIS: Universal Description, Discovery and Integration (UDDI). (2001) http://www.uddi.org/.
6. Risse, T., Knežević, P.: Data storage requirements for the service oriented computing. In: SAINT 2003 - Workshop on Service Oriented Computing. (2003) 67–72
7. Agosti, M., Ferro, N., Frommholz, I., Thiel, U.: Annotations in digital libraries and collaboratories – facets, models and usage. In: Proc. 8th European Conference on Research and Advanced Technology for Digital Libraries (ECDL). (2004) To appear.
8. Frommholz, I., Brocks, H., Thiel, U., Neuhold, E., Iannone, L., Semeraro, G., Berardi, M., Ceci, M.: Document-centered collaboration for scholars in the humanities - the COLLATE system. [30] 434–445
9. Agosti, M., Ferro, N.: Annotations: Enriching a Digital Library. [30] 88–100
10. Thiel, U., Everts, A., Lutes, B., Nicolaides, M., Tzeras, K.: Convergent software technologies: The challenge of digital libraries. In: Proceedings of the 1st Conference on Digital Libraries: The Present and Future in Digital Libraries, Seoul, Korea (1998) 13–30
11. Attardi, G., Gullí, A., Sebastiani, F.: Automatic Web page categorization by link and context analysis. In Hutchison, C., Lanzarone, G., eds.: Proceedings of THAI-99, 1st European Symposium on Telematics, Hypermedia and Artificial Intelligence, (Varese, IT)

12. Neuhold, E.J., Niederée, C., Stewart, A.: Personalization in digital libraries: An extended view. In: Proceedings of ICADL 2003. (2003) 1–16
13. Niederée, C., Stewart, A., Mehta, B., Hemmje, M.: A multi-dimensional, unified user model for cross-system personalization. In: Proceedings of Advanced Visual Interfaces International Working Conference (AVI 2004) - Workshop on Environments for Personalized Information Access, Gallipoli (Lecce), Italy, May 2004. (2004)
14. Özsu, M.T., Valduriez, P.: Principles of Distributed Database Systems. Prentice Hall (1999)
15. Bernstein, P.A., Hadzilacos, V., Goodman, N.: Concurency Control and Recovery in Database Systems. Addison-Wesley (1997)
16. Brunie, L., Kosch, H.: A communications-oriented methodology for load balancing in parallel relational query processing. In: Advances in Parallel Computing, ParCo Conferences, Gent, Belgium. (1995)
17. GPU: A gnutella processing unit (2004) `http://gpu.sf.net`.
18. Müller, W., Henrich, A.: Fast retrieval of high-dimensional feature vectors in P2P networks using compact peer data summaries. In: Proceedings of the 5th ACM SIGMM international workshop on Multimedia information retrieval, ACM Press (2003) 79–86
19. Ovsiannikov, I.A., Arbib, M.A., McNeill, T.H.: Annotation technology. Int. J. Hum.-Comput. Stud. **50** (1999) 329–362
20. Denoue, L., Vignollet, L.: An annotation tool for web browsers and its applications to information retrieval. In: Proceedings of RIAO 2000, Paris, April 2000. (2000)
21. Golovchinsky, G., Price, M.N., Schilit, B.N.: From reading to retrieval: Freeform ink annotations as queries. In Gey, F., Hearst, M., Tong, R., eds.: Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, New York, ACM Press (1999) 19–25
22. Bouthors, V., Dedieu, O.: Pharos, a collaborative infrastructure for web knowledge sharing. In Abiteboul, S., Vercoustre, A.M., eds.: Research and Advanced Technology for Digital Libraries, Proceedings of the Third European Conference, ECDL'99, Paris, France, September 1999. Volume LNCS 1696 of Lecture Notes in Computer Science., Springer-Verlag (1999) 215 ff.
23. McTear, M.: User modeling for adaptive computer systems: A survey of recent developments. In: Artificial Intelligence Review. Volume 7. (1993) 157–184
24. Kaplan, C., Fenwick, J., Chen, J.: Adaptive hypertext navigation based on user goals and context. In: User Modeling and User-Adapted Interaction 3. Kluwer Academic Publishers, The Netherlands (1993) 193–220
25. Goker, A., Myrhaug, H.: User context and personalization. In: Proceedings of the European Conference on Case Based Reasoning (ECCBR 2002) - Workshop on Personalized Case-Based Reasoning, Aberdeen, Scotland, 4-7 September 2002. Volume LNCS 2416 of Lecture Notes in Artificial Intelligence., Springer-Verlag (2002)
26. Pretschner, A., Gauch, S.: Personalization on the web. Technical Report ITTC-FY2000-TR-13591-01, Information and Telecommunication Technology Center (ITTC), The University of Kansas, Lawrence, KS (1999)
27. Gulla, J.A., van der Vos, B., Thiel, U.: An abductive, linguistic approach to model retrieval. Data & Knowledge Engineering **23** (1997) 17–31
28. Casasola, E.: Profusion personalassistant: An agent for personalized information filtering on the www. Master's thesis, The University of Kansas, Lawrence, KS (1998)
29. Meng, X., Chen, Z.: Personalize web search using information on client's side. In: Proceedings of the Fifth International Conference of Young Computer Scientists, August 17-20, 1999, Nanjing, P.R.China, International Academic Publishers (1999) 985–992
30. Koch, T., Sølvberg, I.T., eds.: Proc. 7th European Conference on Research and Advanced Technology for Digital Libraries (ECDL), Lecture Notes in Computer Science (LNCS) 2769, Springer, Heidelberg, Germany (2003)

Towards Collaborative Search in Digital Libraries Using Peer-to-Peer Technology

Matthias Bender, Sebastian Michel, Christian Zimmer, Gerhard Weikum
{mbender, smichel, czimmer, weikum}@mpi-sb.mpg.de

Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany

Abstract. We consider the problem of collaborative search across a large number of digital libraries and query routing strategies in a peer-to-peer (P2P) environment. Both digital libraries and users are equally viewed as peers and, thus, as part of the P2P network. Our system provides a versatile platform for a scalable search engine combining local index structures of autonomous peers with a global directory based on a distributed hash table (DHT) as an overlay network.

1 Introduction

The peer-to-peer (P2P) approach, which has become popular in the context of file-sharing systems such as Gnutella or KaZaA, allows to handle huge amounts of data in a distributed way. In such a system, all peers are equal and all of the functionality is shared among all peers so that there is no single point of failure and the load is balanced across a large number of peers. These characteristics offer potential benefits for building a powerful search engine in terms of scalability, resilience to failures, and high dynamics. In addition, a P2P search engine can potentially benefit from the intellectual input of a large user community, for example, prior usage statistics, personal bookmarks, or implicit feedback derived from user logs and click streams.

Our framework combines well-studied search strategies with new aspects of P2P routing strategies. In our context of digital libraries, a peer can either be a library itself or a user that wants to benefit from the huge amount of data in the network. Each peer is a priori autonomous and has its own local search engine with a crawler and a corresponding local index. Peers share their local indexes (or specific fragments of local indexes) by posting the meta-information into the P2P network, thus effectively forming a large global, but completely decentralized directory. In our approach, this directory is maintained as a distributed hash table (DHT). A query posed by a user is first executed on the user's own peer, but can be forwarded to other peers for better result quality. Collaborative search strategies use the global directory to identify peers that are most likely to hold relevant results. The query is then forwarded to an appropriately selected subset of these peers, and the local results obtained from there are merged by the query initiator.

2 Related Work

Recent research on P2P systems, such as Chord [25], CAN [22], Pastry [24], or P-Grid [1], is based on various forms of distributed hash tables (DHTs) and supports mappings from keys, e.g., titles or authors, to locations in a decentralized manner such that routing scales well with the number of peers in the system. Typically, an exact-match key lookup can be routed to the proper peer(s) in at most $O(log\ n)$ hops, and no peer needs to maintain more than $O(log\ n)$ routing information. These architectures can also cope well with failures and the high dynamics of a P2P system as peers join or leave the system at a high rate and in an unpredictable manner. Earlier work on scalable distributed storage structures, e.g., [16, 28], addressed similar issues. However, in all these approaches searching is limited to exact-match queries on keys. This is insufficient for text queries that consist of a variable number of keywords, and it is absolutely inappropriate when queries should return a ranked result list of the most relevant approximate matches [6]. Our work makes use of one of these systems, namely Chord, for efficiently organizing a distributed global directory; our search engine is layered on top of this basic functionality.

PlanetP [10] is a publish-subscribe service for P2P communities and the first system supporting content ranking search. PlanetP distinguishes local indexes and a global index to describe all peers and their shared information. The global index is replicated using a gossiping algorithm. The system, however, is limited to a few thousand peers.

Odissea [26] assumes a two-layered search engine architecture with a global index structure distributed over the nodes in the system. A single node holds the entire index for a particular text term (i.e., keyword or word stem). Query execution uses a distributed version of Fagin's threshold algorithm [11]. The system appears to cause high network traffic when posting document metadata into the network, and the query execution method presented currently seems limited to queries with one or two keywords only.

The system outlined in [23] uses a fully distributed inverted text index, in which every participant is responsible for a specific subset of terms and manages the respective index structures. Particular emphasis is put on three techniques to minimize the bandwidth used during multi-leyword searches: Bloom filters [3], caching, and incremental result gathering. Bloom filters are a compact representation of membership in a set, eliminating the need to send entire index lists across servers. Caching reduces the frequency of exchanging Bloom filters between servers. Incremental result gathering allows search operations to halt after finding a certain number of results.

[18] considers content-based retrieval in hybrid P2P networks where a peer can either be a simple node or a directory node. Directory nodes serve as superpeers, which may possibly limit the scalability and self-organization of the overall system. The peer selection for forwarding queries is based on the Kullback-Leibler divergence between peer-specific statistical models of term distributions. The approach that we propose in this paper also uses such statistical measures but applies them in a much more light-weight manner for better scalability,

primarily using bookmarks rather than full index information and building on a completely decentralized directory for meta-information.

Strategies for P2P request routing beyond simple key lookups but without considerations on ranked retrieval have been discussed in [30, 8, 7], but are not directly applicable to our setting. The construction of semantic overlay networks is addressed in [17, 9] using clustering and classification techniques; these techniques would be orthogonal to our approach. [27] distributes a global index onto peers using LSI dimensions and the CAN distributed hash table. In this approach peers give up their autonomy and must collaborate for queries whose dimensions are spread across different peers.

In addition to this recent work on P2P Web search, prior research on distributed IR and metasearch engines is potentially relevant, too. [4] gives an overview of algorithms for distributed IR like result merging and database content discovery. [12] presents a formal decision model for database selection in networked IR. [21] investigates different quality measures for database selection. [13, 19] study scalability issues for a distributed term index. GlOSS [14] and CORI [5] are the most prominent distributed IR systems, but neither of them aimed at very-large-scale, highly dynamic, self-organizing P2P environments (which were not an issue at the time these systems were developed).

A good overview of metasearch techniques is given by [20]. [29] discusses specific strategies to determine potentially useful local search engines for a given user query. Notwithstanding the relevance of this prior work, collaborative P2P search is substantially more challenging than metasearch or distributed IR over a small federation of sources such as digital libraries, as these approaches mediate only a small and rather static set of underlying engines, as opposed to the high dynamics of a P2P system.

3 Chord - A Scalable P2P Lookup Service

The efficient location of nodes in a P2P architecture is a fundamental problem that has been tackled from various directions. Early (but nevertheless popular) systems like Gnutella or KaZaA rely on unstructured architectures in which a peer forwards messages to all known neighbors. Typically, these messages include a Time-to-live (TTL) tag that is decreased whenever the message is forwarded to another peer. Even though studies show that this *message flooding* (or *gossiping*) works remarkably well in most cases, there are no guarantees that all relevant nodes will eventually be reached. Additionally, the fact that numerous unnecessary messages are sent interferes with our goal of a highly scalable architecture.

Chord [25] is a distributed lookup protocol that addresses this problem. It provides the functionality of a distributed hash table (DHT) by supporting the following *lookup* operation: given a key, it maps the key onto a node. For this purpose, Chord uses consistent hashing [15]. Consistent hashing tends to balance load, since each node receives roughly the same number of keys. Moreover, this load balancing works even in the presence of a dynamically changing hash range, i.e., when nodes fail or leave the system or when new nodes join.

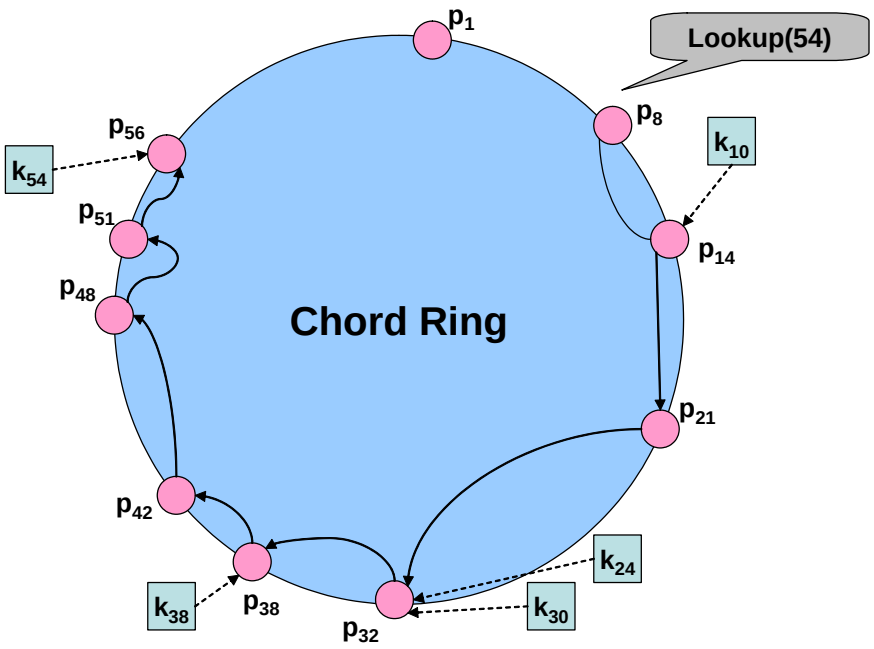


Figure 1. Chord Architecture

Chord not only gurarantees to find the node responsible for a given key, but also can do this very efficiently: in an N-node steady-state system, each node maintains information about only $O(\log N)$ other nodes, and resolves all lookups via $O(\log N)$ messages to other nodes. These properties offer the potential for efficient large-scale systems.

The intuitive concept behind Chord is as follows: all nodes p_i and all keys k_i are mapped onto the same cyclic ID space. In the following, we use keys and peer numbers as if the hash function had already been applied, but we do not explicitly show the hash function for simpler presentation. Every key k_i is now assigned to its closest successor p_i in the ID space, i.e. every node is responsible for all keys with identifiers between the ID of its predecessor node and its own ID.

For example, consider Figure 1. Ten nodes are distributed across the ID space. Key k_{54}, for example, is assigned to node p_{56} as its closest successor node. A naive approach of locating the peer responsible for the key is also illustrated: since every peer knows how to contact its current successor on the ID circle, a query for a key k_{54} initiated by peer p_8 is passed around the circle until it encounters a pair of nodes that straddle the desired identifier; the second in the pair (p_{56}) is the node that is responsible for the key. This lookup process closely resembles searching a linear list and has an expected number of hops of $O(N)$ to find a target node, while only requiring $O(1)$ information about other nodes.

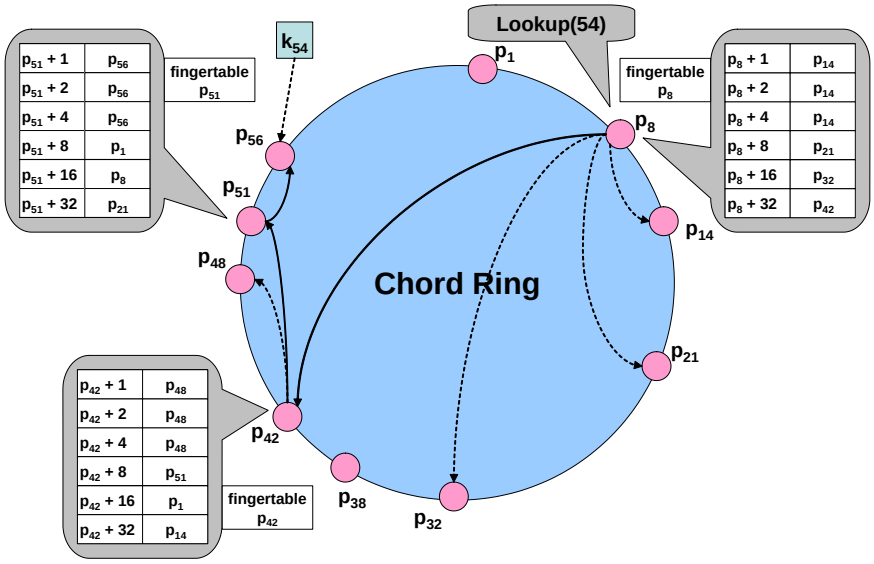


Figure 2. Scalabe Lookups Using Finger Tables

To accelerate lookups, Chord maintains additional routing information: each peer p_i maintains a routing table called *finger table*. The m-th entry in the table of node p_i contains a pointer to the first node p_j that succeeds p_i by at least 2^{m-1} on the identifier circle. This scheme has two important characteristics. First, each node stores information about only a small number of other nodes, and knows more about nodes closely following it on the identifier circle than about nodes farther away. Secondly, a node's finger table does not necessarily contain enough information to *directly* determine the node responsible for an arbitrary key k_i. However, since each peer has finger entries at power of two intervals around the identifier circle, each node can forward a query at least halfway along the remaining distance between itself and the target node. This property is illustrated in Figure 2 for node p_8. It follows that the number of nodes to be contacted (and thus the number of messages to be sent) to find a target node in an N-node system is $O(\log N)$.

Chord implements a stabilization protocol that each peers runs periodically in the background and which updates Chord's finger tables and successor pointers in order to ensure that lookups execute correctly as the set of participating peers changes. But even with routing information becoming stale, system performance degrades gracefully. Chord can also guarantee correct lookups if only one piece of information per node is correct.

Chord can provide lookup services for various applications, such as distributed file systems or cooperative mirroring. However, Chord by itself is not a search engine, as it only supports single-term exact-match queries and does not support any form of ranking.

4 Design Fundamentals

Figure 3 illustrates our new approach which closely follows a publish-subscribe paradigm. We view every library as autonomous. Peers, i.e. libraries acting as peers, can post meta-information at their discretion. Our conceptually global but physically distributed directory does not hold information about individual documents previously crawled by the peers, but only very compact aggregated information about the peers' local indexes and only to the extent that the individual peers are willing to disclose to other peers. We use a distributed hash table (DHT) to partition the term space, such that every peer is responsible for a randomized subset of terms within the global directory. For failure resilience and availability, the entry for a term may be replicated across multiple peers.

Every peer publishes a summary (*Post*) for every term in its local index to the underlying overlay network. A Post is routed to the peer currently responsible for the Post's term. This peer maintains a *PeerList* of all postings for this term from across the network. Posts contain contact information about the peer who posted this summary together with local IR-style statistics (e.g., TF and IDF values [6]) for a term and other quality-of-service measures (e.g., length of the index list for a given term, or average response time for remote queries).

Users wishing to pose a query are equally modelled as peers. Their potential input to the global directory consists of local bookmarks that conceptually represent high-authority documents within the overall document space.

The querying process for a multi-term query proceeds as follows: First, the querying peer retrieves a list of potentially useful libraries by issuing a *PeerList request* for each query term to the global directory. Next, a number of promising libraries for the complete query is selected from these PeerLists (e.g., based on the quality-of-service measures associated with the Posts). Subsequently, the query is forwarded to these carefully selected libraries and executed based on the their local indexes. Note that this communication is done in a pairwise point-to-point manner between the peers, allowing for efficient communication and limiting the load on the global directory. Finally, the results from the various libraries are combined at the querying peer into a single result list.

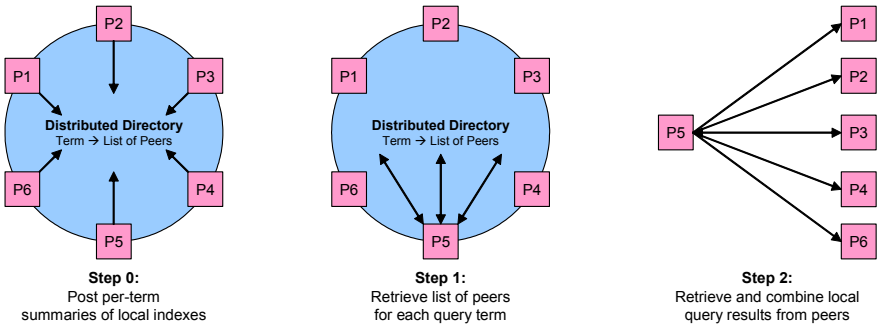


Figure 3. P2P Query Routing

We have chosen this approach for the following reasons:

- The goal of finding high-quality search results with respect to precision and recall cannot easily be reconciled with the design goal of unlimited scalability, as the best information retrieval techniques for query execution rely on large amounts of document metadata. In contrast, posting only aggregated information about local indexes and executing queries at carefully selected peers exploits extensive local indexes for good query results while, at the same time, limiting the size of the global directory and, thus, consuming only little network bandwidth.
- If each peer were to post metadata about each and every document it has crawled, the amount of data moved across the network and, thus, the amount of data held by the distributed directory would increase drastically as more and more peers enter the network. In contrast, our design allows each peer to publish merely a concise summary per term representing its local index. As new peers enter the network, we expect this approach to scale very well as more and more peers jointly maintain this moderately growing global directory.

This approach can easily be extended in a way that multiple distributed directories are created to store information beyond local index summaries, such as information about user bookmarks or relevance assessments derived from peer-specific query logs, click streams, or explicit user feedback. This information could be leveraged when executing a query to further enhance result quality.

5 System Model

In this section we formalize the design that we have previouly presented. Let $P := \{p_i | 1 \leq i \leq r\}$ be the set of peers currently attached to the system. Let $D := \{d_i | 1 \leq i \leq n\}$ be the global set of all documents; let $T := \{t_i | 1 \leq i \leq m\}$ analogously be the set of all terms.

Each peer $p_i \in P$ has one or more of the following local data available:

- Local index lists for terms in $T_i \subseteq T$ (usually $|T_i| \ll |T|$).
  The local index lists cover all terms in the set of locally seen documents $D_i \subseteq D$ (usually $|D_i| \ll |D|$).
- Bookmarks $B_i \subseteq D_i$ ($|B_i| \ll |D|$)
  Bookmarks are intellectually selected links to selected documents or other peer profile information and, thus, are a valuable source for high-quality search results as well as for the thematic classification of peers.
- Cached documents $C_i \subseteq D$
  Cached documents are readily available from a peer.

Separate hash functions $hash_{terms} : T \rightarrow ID$, $hash_{bookmarks} : D \rightarrow ID$, and $hash_{cached} : D \rightarrow ID$ can be used in order to build conceptually global, but physically distributed directories that are well-balanced across the peers in the ID space.

Given hash functions that assign identifiers to keys using $id_{k,j} := hash_j(k)$ with $j \in \{terms, bookmarks, ...\}$, the underlying distributed hash table offers a function $lookup : ID \rightarrow P$ that returns the peer p currently responsible for an id.

Building on top of this basic functionality, different PeerList requests plr_j can be defined as functions $plr_{terms} : T \times P \rightarrow 2^P$, $plr_{bookmarks} : D \times P \rightarrow 2^P$, and $plr_{cache} : D \times P \rightarrow 2^P$ that, from a peer p previously determinded using *lookup*, return lists of peers that have posted information about a key id in dimension j. Note that $id_{k,j}$ for a specific key k is unambiguously defined across the directory using $hash_j$(k).

In order to form a distributed directory, each peer p_i at its own discretion globally posts subsets $T_i' \subseteq T_i$, $B_i' \subseteq B_i$, and $C_i' \subseteq C_i \subseteq D$

(potentially along with further information or local QoS statistics) forming the corresponding global directories:

- $systerms : T \rightarrow 2^P$ with $systerms(t) = plr_{terms}(t, lookup(hash_{terms}(t)))$
  This directory provides a mapping from terms to PeerLists and can be used to identify candidate peers that hold index information about a specific term.
- $sysbm : D \rightarrow 2^P$ with $sysbm(d) = plr_{bookmarks}(d, lookup(hash_{bookmarks}(d)))$
  This function provides information about which peers have bookmarked specific documents and is a combination of the above methods analogously to *systerms*.
- $syscd : D \rightarrow 2^P$ with $syscd(d) = plr_{cached}(d, lookup(hash_{cached}(d)))$
  This function provides information about the availability of documents in the caches of local peers, which is a valuable information for the efficient gathering of results.

We consider a query q as a set of $(term, weight)$-pairs and the set of available queries as $Q := 2^{T \times \mathbb{R}}$. In order to process a query q, first a candidate set of peers that are confronted with the query has to be determined. This can be done using the functions $selection_{terms} : Q \rightarrow 2^P$, $selection_{bookmarks} : 2^D \rightarrow 2^P$, and $selection_{cached} : 2^D \rightarrow 2^P$ that select candidate subsets for each dimension by appropriately combining the results returned by *systerms*, *sysbm*, and *syscd*, respectively. These candidate subsets are combined (e.g., by intersection or union) using a function $comb : 2^P \times 2^P \times 2^P \rightarrow 2^P$.

Combining the above, the final candidate set is computed using a function

$$selection : Q \times 2^D \times 2^D \rightarrow 2^P$$

$$selection(q, B_0'', C_0'') := comb(select_{terms}(q), select_{bookmarks}(B_0''), select_{cached}(C_0''))$$

where $B_0'' \subseteq B_0$ and $C_0'' \subseteq C_0$ are the bookmarks and cached documents, respectively, that the querying peer has chosend to support query execution. For example, a peer may choose its own bookmarks and a sample of its cached documents as B_0'' and C_0'', respectively.

The execution of a query is a function $exec : 2^P \times Q \rightarrow 2^D$ that combines the local results returned by the peers that are involved in the query execution into one single final result set. Finally, we can define the global query execution function $result : Q \times 2^D \times 2^D \rightarrow 2^D$ that is evaluated as

$$result(q, B_0'', C_0'') := exec(selection(q, B_0'', C_0''), q)$$

$$= exec(comb(select_{terms}(q), select_{bookmarks}(B_0''), select_{cached}(C_0'')), q)$$

6 Implementation

Figure 4 illustrates the architecture of a single library peer as part of our distributed system. Each peer works on top of our globally distributed index which is organized as a distributed hash table (DHT) that provides a mapping from terms to peers by returning a *PeerDescriptor* object representing the peer currently responsible for a term. A *Communicator* can be established to send messages to other peers. Every peer has an *Event Handler* that receives incoming messages and forwards them to the appropriate local components.

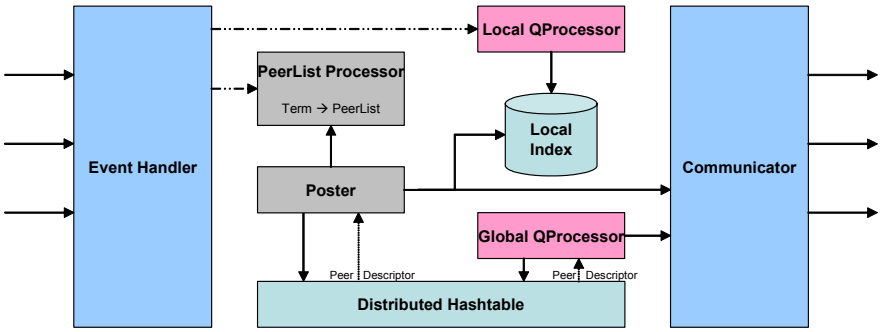


Figure 4. System Architecture

Every peer has its own local index that can be imported from external crawlers and indexers. The index is used by the *Local QueryProcessor* component to answer queries locally and by the *Poster* component to publish per-term summaries (*Posts)* to the global directory. To do so, the Poster uses the underlying DHT to find the peer currently responsible for a term; the *PeerList Processor* at this peer maintains a PeerList of all Posts for this term from across the network. When the user poses a query, the *Global QueryProcessor* component analogously uses the DHT to find the peer responsible for each query term and retrieves the respective PeerLists from the PeerList Processors using Communicator components. After appropriately processing these lists, the Global QueryProcessor forwards the complete query to selected peers, which in turn process the query using their Local QueryProcessors and return their results. Finally, the Global QueryProcessor merges these results and presents them to the user.

We have built a prototype system that handles the above procedures. Our system uses a Java-based reimplementation of Chord [25] as its underlying DHT, but can easily be used with other DHT's providing a *lookup*(*key*) method. Communication is conducted socket-based, but Web-Service-based [2] peers can easily be included to support an arbitrarily heterogeneous environment. The local index is stored in a database. It consists of a collection of standard IR measures, such as TF and IDF values. Result ranking is based on a smoothed TF*IDF quality measure. Figure 5 shows a screenshot of the user interface of our prototype. The user creates a peer by either creating a new Chord ring or by joining an existing

system. Both actions require the specification of a local Chord port for communication concerning the global directory and a local application port for direct peer-to-peer communication. The join operation requires additional information on how to find an already existing peer. Status information regarding the Chord ring is displayed. The Posts section provides information about the terms that a peer is currently responsible for, i.e., for which it has received Posts from other peers. The button *Post* posts the information contained in the local index to the DHT. The Queries section can be used to execute queries. Similar to Google, multiple keywords can be entered into a form field. After query execution, the results obtained from the system are displayed.

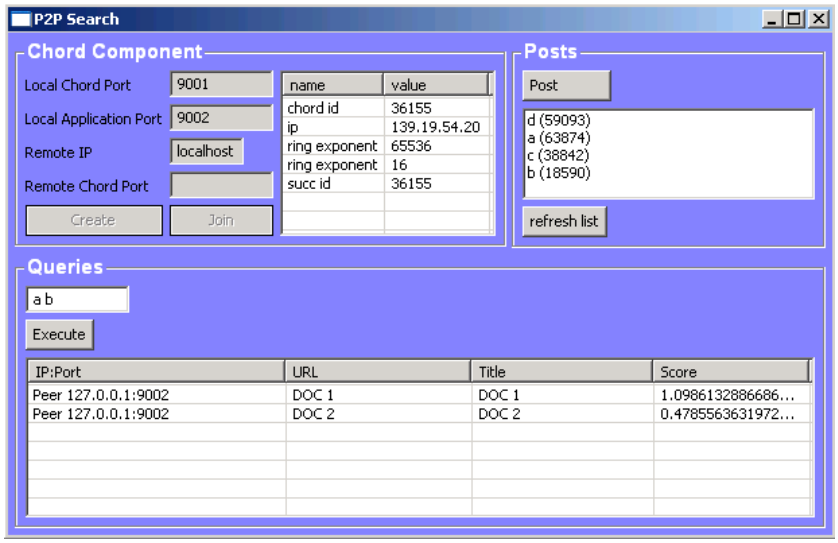


Figure 5. Prototype GUI

7 Ongoing and Future Work

Our prototype implementations allows for the easy exchange of strategies for query routing (i.e., selecting the peers to which the query is sent) as well as for merging the results returned by different peers. We are currently analyzing different strategies and are preparing extensive comparative experiments. We want to contact as few peers as possible to retrieve the best possible results, i.e., we want to estimate a $benefit/cost$ ratio when deciding on whether to contact a specific peer. While a typical cost measure could be based on expected response time (network latency, current load of remote peer), meaningful benefit measures seem harder to find. Possible measures could follow the intuition that good answers are expected to come from peers that are similar to the query, but at the same time have only little overlap with our local index and, thus, can potentially contribute new results. We also take a closer look at existing

strategies for combining local query results from metasearch engine research and try to fit those with our P2P environment.

We investigate the trade-offs of not storing *all* Posts for a term, but only the top-k posts (based on some quality measure) to reduce space consumption of the global directory. While this seems intuitive at first sight (good results should come from good peers), early experiments indicate that this strategy might be dangerous for multi-term queries, as good combined results are not necessarily top results for any one of the search terms.

Due to the dynamics typical for P2P systems, Posts stored in the PeerLists become invalid (peers may no longer be accessible, or the responsibility for a specific term may have moved to another peer). A possible mechanism to handle these problems is to assign a TTL (Time-to-live) stamp to every Post in the list. Every peer periodically revalidates its Posts. Stale Posts will eventually be removed from the PeerList. We address the question of choosing a good time period for refreshing the Posts and compare this strategy to a strategy of actively moving Posts to other peers as responsibilities change.

References

1. K. Aberer, M. Punceva, M. Hauswirth, and R. Schmidt. Improving data access in p2p systems. *IEEE Internet Computing*, 6(1):58–67, 2002.
2. G. Alonso, F. Casati, and H. Kuno. *Web Services - Concepts, Architectures and Applications*. Springer, Berlin;Heidelberg;New York, 2004.
3. B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.
4. J. Callan. Distributed information retrieval. *Advances in information retrieval, Kluwer Academic Publishers.*, pages 127–150, 2000.
5. J. P. Callan, Z. Lu, and W. B. Croft. Searching distributed collections with inference networks. In *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 21–28. ACM Press, 1995.
6. S. Chakrabarti. *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan Kaufmann, San Francisco, 2002.
7. E. Cohen, A. Fiat, and H. Kaplan. Associative search in peer to peer networks: Harnessing latent semantics. In *Proceedings of the IEEE INFOCOM'03 Conference, April 2003*, April 2003.
8. A. Crespo and H. Garcia-Molina. Routing indices for peer-to-peer systems. In *Proc. of the 28th Conference on Distributed Computing Systems*, July 2002.
9. A. Crespo and H. Garcia-Molina. Semantic Overlay Networks for P2P Systems. Technical report, Stanford University, October 2002.
10. F. M. Cuenca-Acuna, C. Peery, R. P. Martin, and T. D. Nguyen. PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities. Technical Report DCS-TR-487, Rutgers University, Sept. 2002.
11. R. Fagin. Combining fuzzy information from multiple systems. *J. Comput. Syst. Sci.*, 58(1):83–99, 1999.
12. N. Fuhr. A decision-theoretic approach to database selection in networked IR. *ACM Transactions on Information Systems*, 17(3):229–249, 1999.
13. T. Grabs, K. Böhm, and H.-J. Schek. Powerdb-ir: information retrieval on top of a database cluster. In *Proceedings of the tenth international conference on Information and knowledge management*, pages 411–418. ACM Press, 2001.

14. L. Gravano, H. Garcia-Molina, and A. Tomasic. Gloss: text-source discovery over the internet. *ACM Trans. Database Syst.*, 24(2):229–264, 1999.
15. D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *ACM Symposium on Theory of Computing*, pages 654–663, May 1997.
16. W. Litwin, M.-A. Neimat, and D. A. Schneider. Lh* – a scalable, distributed data structure. *ACM Trans. Database Syst.*, 21(4):480–525, 1996.
17. A. Löser, F. Naumann, W. Siberski, W. Nejdl, and U. Thaden. Semantic overlay clusters within super-peer networks. In *Proceedings of the International Workshop on Databases, Information Systems and Peer-to-Peer Computing, 2003 (DBISP2P 03)*, pages 33–47.
18. J. Lu and J. Callan. Content-based retrieval in hybrid peer-to-peer networks. In *Proceedings of the twelfth international conference on Information and knowledge management*, pages 199–206. ACM Press, 2003.
19. S. Melnik, S. Raghavan, B. Yang, and H. Garcia-Molina. Building a distributed full-text index for the web. *ACM Trans. Inf. Syst.*, 19(3):217–241, 2001.
20. W. Meng, C. T. Yu, and K.-L. Liu. Building efficient and effective metasearch engines. *ACM Computing Surveys*, 34(1):48–89, 2002.
21. H. Nottelmann and N. Fuhr. Evaluating different methods of estimating retrieval quality for resource selection. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 290–297. ACM Press, 2003.
22. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM 2001*, pages 161–172. ACM Press, 2001.
23. P. Reynolds and A. Vahdat. Efficient peer-to-peer keyword searching. In *Proceedings of International Middleware Conference*, pages 21–40, June 2003.
24. A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, 2001.
25. I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications*, pages 149–160. ACM Press, 2001.
26. T. Suel, C. Mathur, J. Wu, J. Zhang, A. Delis, M. Kharrazi, X. Long, and K. Shanmugasunderam. Odissea: A peer-to-peer architecture for scalable web search and information retrieval. Technical report, Polytechnic Univ., 2003.
27. C. Tang, Z. Xu, and S. Dwarkadas. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 175–186. ACM Press, 2003.
28. R. Vingralek, Y. Breitbart, and G. Weikum. Snowball: Scalable storage on networks of workstations with balanced load. *Distributed and Parallel Databases*, 6(2):117–156, 1998.
29. Z. Wu, W. Meng, C. T. Yu, and Z. Li. Towards a highly-scalable and effective metasearch engine. In *World Wide Web*, pages 386–395, 2001.
30. B. Yang and H. Garcia-Molina. Improving search in peer-to-peer networks. In *Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, pages 5–14. IEEE Computer Society, 2002.

Web Services for Peer-to-Peer Resource Discovery on the Grid

Domenico Talia and Paolo Trunfio

DEIS, University of Calabria
Via P. Bucci 41c, 87036 Rende, Italy
{talia,trunfio}@deis.unical.it

Abstract. Several features of today's Grids are based on centralized or hierarchical services. However, as Grid sizes increase, some of their functions should be decentralized to avoid bottlenecks and guarantee scalability. A way to provide Grid scalability is to adopt *Peer-to-Peer* (*P2P*) models and protocols to implement non hierarchical decentralized Grid services and systems. A core Grid functionality that could be effectively redesigned using the P2P model is *resource discovery*. This paper proposes an architecture for resource discovery that adopts a P2P approach to extend the model of the Globus Toolkit 3 information service. The *Open Grid Services Architecture* is exploited to define a *P2P Layer* of specialized Grid Services that support resource discovery across different Virtual Organizations in a P2P fashion. The paper discusses also a protocol, named *Gridnut*, designed for communication among Grid Services at the P2P Layer.

1 Introduction

The Grid computing model offers an effective way to build high-performance computing systems, allowing users to efficiently access and integrate geographically distributed computers, data, and applications. Several features of today's Grids are based on centralized or hierarchical services. However, as Grids used for complex applications increase their size from tens to thousands of nodes, it is necessary to decentralize their services to avoid bottlenecks and ensure scalability. As argued in [1] and [2], a way to provide Grid scalability is to adopt *Peer-to-Peer* (*P2P*) models and techniques to implement non-hierarchical decentralized Grid systems.

In the latest years, the Grid community has undertaken a development effort to align Grid technologies with Web Services. The *Open Grid Services Architecture* (*OGSA*) defines *Grid Services* as an extension of Web Services and lets developers integrate services and resources across distributed, heterogeneous, dynamic environments and communities [3]. Web Services define a technique for describing software components to be accessed, methods for accessing these components, and discovery methods that enable the identification of relevant service providers. Web Services and OGSA aim at interoperability between loosely coupled services independently from implementation, location or platform. Recently

the *Web Services Resource Framework* (*WSRF*) has been proposed for a more complete integration between Web and Grid Services [4]. OGSA defines standard mechanisms for creating, naming and discovering persistent and transient Grid Service instances, provides location transparency and multiple protocol bindings for service instances, and supports integration with underlying native platform facilities. The OGSA effort aims to define a common resource model that is an abstract representation of both real resources, such as processors, processes, disks, file systems, and logical resources. It provides some common operations and supports multiple underlying resource models representing resources as service instances. The OGSA model provides an opportunity to integrate P2P models in Grid environments since it offers an open cooperation model that allows Grid entities to be composed in a decentralized way.

A core Grid functionality that could be effectively redesigned using the P2P model is *resource discovery*. Resource discovery is a key issue in Grid environments, since applications are usually constructed by composing hardware and software resources that need to be discovered and selected. In the OGSA framework each resource is represented as a Grid Service, therefore resource discovery mainly deals with the problem of locating and querying information about useful Grid Services.

In *Globus Toolkit 3* (*GT3*) - the current implementation of the OGSA - information about resources is provided by *Index Services*. An Index Service is a Grid Service that holds information (called *Service Data*) about a set of Grid Services registered to it. A primary function of the Index Service is to provide an interface for querying aggregate views of Service Data collected from registered services. There is typically one Index Service per *Virtual Organization* (*VO*). When a VO consists of multiple large sites, very often each site runs its own Index Service that indexes the various resources available at that site. Then each of those Index Services is included in the VO's Index Service [5].

This paper proposes an architecture for resource discovery that adopts a P2P approach to extend the model of the GT3 information service. In particular, a *P2P Layer* of specialized Grid Services is defined to support discovery queries on Index Services of multiple VOs in a P2P fashion. The paper outlines also a modified Gnutella protocol, named *Gridnut*, designed for communication among Grid Services at the P2P Layer. Gridnut uses appropriate message buffering and merging techniques to make Grid Services effective as a way to exchange messages in a P2P fashion.

The remainder of the paper is organized as follows. Section 2 describes the architecture of the framework. Section 3 discusses the Gridnut approach and its performances. Finally, Section 4 concludes the paper.

2 The P2P Architecture

From the perspective of the GT3 information service, the Grid can be seen as a collection of VOs, each one indexed by a different Index Service. As mentioned before, Index Services of different sites can be included in a common higher-

level Index Service that holds information about all the underlying resources. However, for scalability reasons, a multi-level hierarchy of Index Services is not appropriate as a general infrastructure for resource discovery in large scale Grids. Whereas centralized or hierarchical approaches can be efficient to index resources structured in a given VO, they are inadequate to support discovery of resources that span across many independent VOs. The framework described here adopts the P2P model to support resource discovery across different VOs.

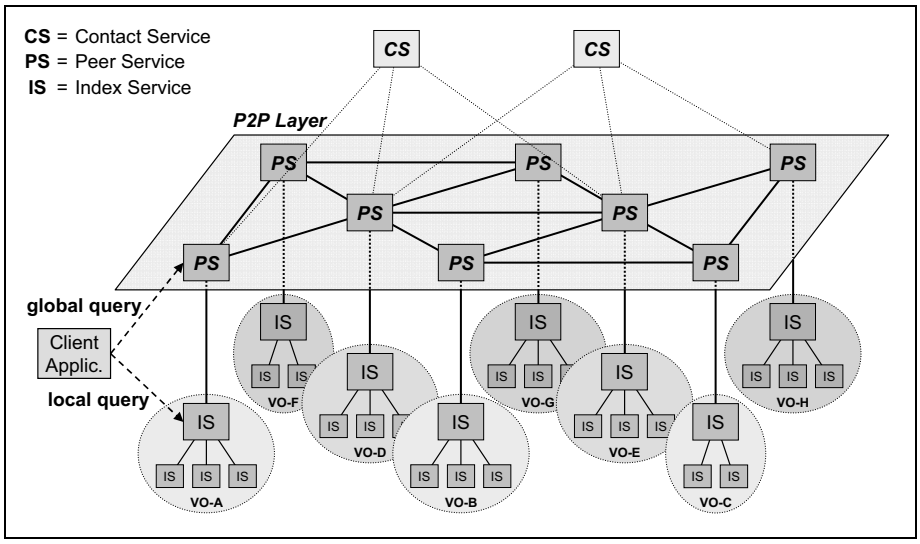


Fig. 1. Framework architecture.

Figure 1 shows the general architecture of the framework. Some independent VOs are represented; each VO provides one top-level *Index Service* (*IS*) and a number of lower-level Index Services.

A *P2P Layer* is defined on top of the Index Services' hierarchy. It includes two types of specialized Grid Services: *Peer Services* (*PS*), used to perform resource discovery, and *Contact Services* (*CS*), that support Peer Services to organize themselves in a P2P network.

There is one Peer Service per VO. Each Peer Service is *connected* with a set of Peer Services, and exchanges query/response messages with them in a P2P mode. The connected Peer Services are the *neighbors* of a Peer Service. A *connection* between two neighbors is a logical state that enables they to directly exchange messages. Direct communication is allowed only between neighbors. Therefore, a query message is sent by a Peer Service only to its neighbors, which in turn will forward that message to their neighbors. A query message is processed by a Peer Service by invoking the top-level Index Service of the corresponding VO.

A query response is sent back along the same path that carried the incoming query message.

To join the P2P network, a Peer Service must know the URL of at least one Peer Services to connect to. A convenient number of Contact Services is distributed in the Grid to support this issue. Contact Services cache the URLs of known Peer Services; a Peer Service may contact one or more well known Contact Services to obtain the URLs of registered Peer Services.

As shown in Figure 1, a *Client Application* can submit both *local* and *global* queries to the framework. A local query searches for information about resources in a given VO. It is performed by submitting the query to the Index Service of that VO. A global query aims to discover resources located in possibly different VOs, and is performed by submitting the query to a Peer Service at the P2P Layer. As mentioned before, the Peer Service processes that query internally (through the associated Index Service), and will forward it to its neighbors as in typical P2P networks.

The main difference between a hierarchical system and the framework described here is the management of global queries. Basically, in a hierarchical information service two alternative approaches can be used:

- the query is sent separately to all the top-level Index Services, that must be known by the user;
- the query is sent to one (possibly replicated) Index Service at the root of the hierarchy, that indexes all the Grid resources.

Both these approaches suffer scalability problems. In the P2P approach, conversely, global queries are managed by a layer of services that cooperate as peers. To submit a global query, a user need only to know the URL of a Peer Service in the Grid.

In the next subsection the design of the Peer Service and Contact Service components is discussed.

2.1 Services Design

Both Peer Service and Contact Service instances are identified by a globally unique *handle*. This handle is a URL - called *grid service handle* (*GSH*) in the OGSA terminology - that distinguishes a specific Grid Service instance from all other Grid Service instances.

The Peer Service supports four operations:

- ***connect***: invoked by a remote Peer Service to connect this Peer Service. The operation receives the *handle* of the requesting Peer Service and returns a *reject* response if the connection is not accepted (for instance, because the maximum number of connections has been reached).
- ***disconnect***: invoked by a remote Peer Service to disconnect this Peer Service. The operation receives the *handle* of the requesting Peer Service.
- ***deliver***: invoked by a connected Peer Service to deliver messages to this Peer Service. The operation receives the *handle* of the requesting Peer Service and an *array of messages* to be delivered to this Peer Service.

– ***query***: invoked by a client application to submit a global *query* to this Peer Service. Query responses are returned to the client through a notification mechanism.

The Contact Service supports one operation:

– ***getHandles***: invoked by a Peer Service to register itself and to get the handles of one or more registered Peer Services.

The *Web Services Description Language* (*WSDL*) is used in OGSA to expose the interfaces of Grid Services to remote clients. Figure 2, for instance, shows the Contact Service WSDL definition.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="ContactService"
 targetNamespace="http://www.gridnut.org/namespaces/1.0/contact/ContactService"
 ... >
...
<gwsdl:portType name="ContactServicePortType" extends="ogsi:GridService">
   <operation name="getHandles">
      <input message="tns:GetHandlesInputMessage"/>
      <output message="tns:GetHandlesOutputMessage"/>
      <fault name="Fault" message="ogsi:FaultMessage"/>
   </operation>
</gwsdl:portType>
<message name="GetHandlesInputMessage">
   <part name="parameters" element="tns:getHandles"/>
</message>
<message name="GetHandlesOutputMessage">
   <part name="parameters" element="tns:getHandlesResponse"/>
</message>
<types>
<xsd:schema
 targetNamespace="http://www.gridnut.org/namespaces/1.0/contact/ContactService"
 ... >
   <xsd:element name="getHandles">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="handle" type="xsd:string"/>
            <xsd:element name="numHandles" type="xsd:int"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="getHandlesResponse">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="getHandlesReturn" type="soapenc:Array"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
</types>
</definitions>
```

Fig. 2. Contact Service WSDL definition.

Figure 3 and Figure 4 describe, respectively, the main software components of Peer Services and Contact Services.

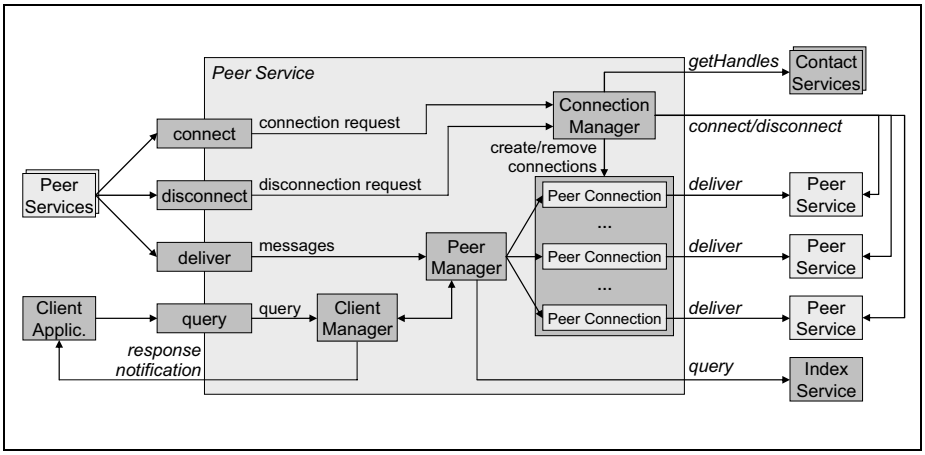


Fig. 3. Peer Service software components.

The Peer Service (see Figure 3) is composed by three main modules: *Connection Manager*, *Peer Manager*, and *Client Manager*.

The goal of the Connection Manager is to maintain a given number of connections with neighbor Peer Services. A *Peer Connection* object is used to manage the connection and the exchange of messages with a given Peer Service. A Peer Connection includes the *grid service reference* (*GSR*) of a given Peer Service, and a set of *transmission buffers* for the different kinds of messages directed to it. The Connection Manager both manages connection/disconnection requests from remote Peer Services, and performs connection/disconnection requests (as a client) to remote Peer Services. Moreover, it may invoke one or more Contact Services to obtain the handles of Peer Services to connect to.

The Peer Manager is the core component of the Peer Service. It both manages the messages delivered from other Peer Services, and interacts with the Client Manager component to manage client requests and to provide responses. It performs different operations on delivered messages: some messages are simply forwarded to one or more Peer Connections, whereas query messages need also a response (that in general is obtained by querying the local Index Service). Moreover, the Peer Manager generates and submits query messages to the network on the basis of the Client Manager requests.

The Client Manager manages the query requests submitted by client applications. It interacts with the Peer Manager component to submit the query to the network, and manages the delivery of query results to the client through a notification mechanism.

The Contact Service (see Figure 3) is composed by two software modules: *Contact Manager* and *Cache Manager*.

The Contact Manager manages the execution of the *getHandles* operation. Basically, it receives two parameters: the handle h of the invoker, and the number

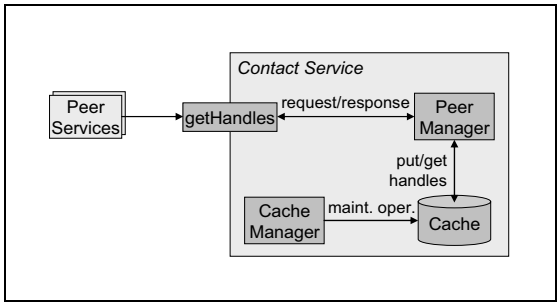


Fig. 4. Contact Service software components.

n of handles requested by the invoker. The Contact Manager first inserts (or updates) the handle h into a *Cache*, then it extracts (if available) n distinct handles from the Cache and returns them to the invoker. The handles can be extracted from the Cache on the basis of a given policy (e.g., randomly). If a Peer Service does not receive the requested number of handles, it can try to invoke the Contact Service later.

The Cache Manager performs maintenance operations on the Cache. For instance, it removes oldest (not recently updated) handles, performs content indexing, etc.

3 A P2P Grid Services-Based Protocol

Although Grid Services are appropriate for implementing loosely coupled P2P applications, they appear to be inefficient to support an intensive exchange of messages among tightly coupled peers. In fact, Grid Services operations are subject to an invocation overhead that can be significant both in terms of activation time and memory/processing consumption [6]. The number of Grid Service operations that a peer can efficiently manage in a given time interval depends strongly on that overhead. For this reason, standard P2P protocols based on a pervasive exchange of messages, such as Gnutella [7], are inappropriate on large OGSA Grids where a high number of communications take place among hosts.

To overcome this limitation, in [8] we proposed a modified Gnutella protocol, named *Gridnut*, which uses appropriate message buffering and merging techniques to make Grid Services effective as a way for exchanging messages among Grid nodes in a P2P fashion. The Gridnut protocol is designed for communication among Peer Services to support resource discovery across many independent VOs.

There are two main differences between Gnutella and Gridnut:

1. In Gnutella, messages are sent as a byte stream over TCP sockets, whereas in Gridnut messages are sent through a Grid Service invocation (by means of the Peer Service's *deliver* operation).

2. In Gnutella, each message is forwarded whenever it is received, whereas in Gridnut messages are buffered and merged to reduce the number of Grid Service invocations and routing operations executed by each Peer Service.

In particular, the basic principles adopted by Gridnut to reduce communication and routing overhead are

- *Message buffering*: messages to be delivered to the same Peer Service are buffered and sent in a single packet at regular time intervals.
- *Message merging*: messages with the same header (i.e, same type, identifier, and receiver) are merged into a single message with a cumulative body.

Similarly to Gnutella, Gridnut defines both a protocol to discover active Peer Services on the network, based on a *Ping/Pong* mechanism, and a protocol for searching the distributed network, based on a *Query/QueryHit* mechanism. We implemented and evaluated the Gridnut discovery protocol, even if we are also designing a general Gridnut search protocol.

In the following subsection we briefly compare the performance of Gridnut and Gnutella discovery protocols under different network and load conditions. Further details about performance measurements can be found in [8].

3.1 Performance Evaluation

The goal of our tests is to verify how significantly Gridnut reduces the workload - number of Grid Service operations - of each Peer Service. In doing this, we compared Gridnut and Gnutella by evaluating two parameters:

1. *ND*, the average number of *deliver* operations processed by a Peer Service to complete a discovery task. In particular, $ND = P \ / \ (N * T)$, where: P is the total number of *deliver* operations processed in the network, N is the number of Peer Services in the network, and T is the overall number of discovery tasks completed.
2. $ND(d)$, the average number of *deliver* operations processed by Peer Services that are at distance d from the Peer Service *S0* that started the discovery task. For instance: $ND(0)$ represents the number of *deliver* operations processed by *S0*; $ND(1)$ represents the number of *deliver* operations processed by a Peer Service distant one hop from *S0*.

Both *ND* and $ND(d)$ have been evaluated considering different network topologies. We distinguished the network topologies using a couple of numbers $\{N,C\}$, where N is the number of Peer Services in the network, and C is the number of Peer Services directly connected to each Peer Service (i.e., each Peer Service has exactly C neighbors).

Number of Deliver Operations

Figure 5 compares the values of *ND* in Gridnut and Gnutella in five network topologies: {10,2}, {30,3}, {50,4}, {70,4} and {90,4}, under different load conditions. A parameter R is used to define the load of the network. In particular,

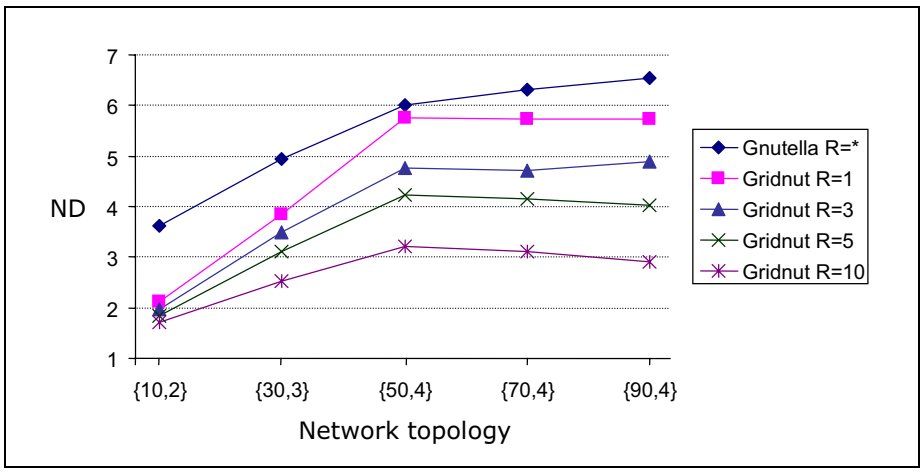


Fig. 5. ND versus the network topology.

R indicates the number of simultaneous discovery tasks that are initiated in the network at each given time interval. For Gridnut networks the values of *ND* when $R = 1$, 3, 5, and 10 are represented, whereas for Gnutella networks the average of the *ND* values measured when $R = 1$, 3, 5, and 10 is represented.

We can see that the number of *deliver* operations is lower with Gridnut in all the considered configurations. In particular, when the load of the network increases, the Gridnut strategy maintains the values of *ND* significantly low in comparison with Gnutella.

Distribution of Deliver Operations

Figure 6 compares the values of $ND(d)$ in Gridnut and Gnutella in five network topologies, with d ranging from *0* to *2*, and R fixed to *1*.

We can see that Gridnut implies a much better distribution of *deliver* operations among Peer Services in comparison with Gnutella. In Gnutella, the Peer Service that started the discovery task and its closest neighbors must process a number of Grid Service operations that becomes unsustainable when the size of the network increases to thousands of nodes. In Gridnut, conversely, the number of Grid Service operations processed by each Peer Service remains always in the order of the number of connections per peer.

This Gridnut behavior results in significantly lower discovery times since communication and computation overhead due to Grid Services invocations are considerably reduced as shown in Figure 6.

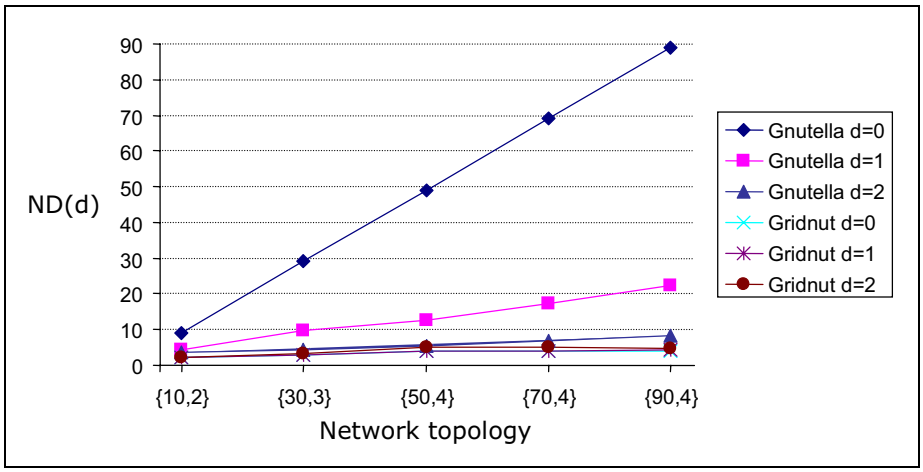


Fig. 6. ND(d) versus the network topology.

4 Conclusions

Resource discovery is a key issue in Grid environments, since applications are usually constructed by composing hardware and software resources that need to be discovered and selected. This paper proposed a framework for resource discovery that adopts a P2P approach to extend the model of the Globus Toolkit 3 information service. It defines a *P2P Layer* of specialized Grid Services that support resource discovery across different Virtual Organizations in a P2P fashion.

The paper discussed also a protocol, named *Gridnut*, designed for communication among Grid Services at the P2P Layer. Gridnut uses message buffering and merging techniques to make Grid Services effective as a way for exchanging messages among Grid nodes in a P2P mode. Experimental results show that appropriate message buffering and merging strategies produce significant performance improvements, both in terms of number and distribution of Grid Service operations processed.

Acknowledgements

This work has been partially funded by the Italian MIUR project “Grid.it” (RBNE01KNFP).

References

1. Foster, I., Iamnitchi, A.: On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. Proc. of the 2nd International Workshop on Peer-to-Peer Systems (2003)

2. Talia, D., Trunfio, P.: Toward a Synergy between P2P and Grids. IEEE Internet Computing, vol. 7, n. 4 (2003) 94-96
3. Foster, I., Kesselman, C., Nick, J. M., Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. http://www.globus.org/research/papers/ogsa.pdf
4. Czajkowski, K. et al.: From Open Grid Services Infrastructure to WS-Resource Framework: Refactoring & Evolution. http://www-106.ibm.com/developerworks/library/ws-resource/ogsi_to_wsrf_1.0.pdf
5. The Globus Alliance: MDS Functionality in GT3. http://www.globus.org/ogsa/releases/final/docs/infosvcs/4.html
6. The Globus Alliance: Globus Toolkit 3.0 - Performance Tuning Guide. http://www-unix.globus.org/toolkit/3.0/ogsa/docs/performance_guide.html
7. Clip2: The Gnutella Protocol Specification v.0.4. http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf
8. Talia, D., Trunfio, P.: A P2P Grid Services-Based Protocol: Design and Evaluation. Proc. of the European Conference on Parallel Computing - EuroPar 2004 - (in press)

Collaboration of loosely coupled repositories using peer-to-peer paradigm

András Micsik[1], László Kovács[1], Robert Stachel[2]

[1]MTA SZTAKI DSD,
Lágymányosi utca 11, H-1111 Budapest, Hungary
{micsik,laszlo.kovacs}@sztaki.hu
[2]Team Teichenberg,
Operngasse 22, A-1050 Vienna, Austria
robert@teichenberg.at

Abstract. A distributed digital library has been designed and implemented for the support of community radios. This framework, developed by the StreamOnTheFly IST project of the EU, provides a common background for preparation, archival, exchange and reuse of radio programmes and supports radio personalization. The architecture is based on a decentralized and self-organizing network, which uses a new common metadata schema and exchange format and automatic metadata replication. Local, regional and Internet-based radios are expected to join the network all over Europe.

1 Introduction

Nowadays many community radio stations (non-profit, independent stations) exist, but there is little cooperation among those. One of the causes for this is the lack of technical support for easy exchange of radio programmes. A significant part of the programmes produced at community radio stations are of high quality and worth to be broadcast more than once, and listened to by a larger number of people in a wider geographical range.

The StreamOnTheFly IST project [7] was set out to find and demonstrate new ways for archival, management and personalization of audio content. Within the project Public Voice Lab (Austria) and MTA SZTAKI DSD (Department of Distributed Systems at the Computer and Automation Research Institute of the Hungarian Academy of Sciences) are development partners and Team Teichenberg (Austria) provides connections to community radio stations and associations.

The project team envisioned a network of digital archives and helper tools for radio stations. The general aim of this network was to foster alternative, self-organized and independent media by new ways of content distribution, international community based presentation platforms and many added values for local production and organizational work. In the following we will give an overview on the world of radio, describe self-organizing systems on the Internet, present the StreamOnTheFly network and the work done so far.

2 Community Radios in Europe

Community radio stations developed during the past 30 years in most countries of Europe, enriching the existing media world with a "third sector", next to public broadcasting as the first and commercial media as the second. Community radios are non-profit-oriented, open to the general population and have a local or regional range. They provide access to radio production facilities for organizations, groups and individuals who aim to make their own radio shows. There are over 1500 non-profit stations in the whole of Europe, an estimated third of them is provided with a full and permanent 24 hour broadcasting license on their own frequency.

The global umbrella organization AMARC defines community radio as *"which is for, by and about the community, whose ownership and management is representative of the community, which pursues a social development agenda, and which is non-profit."* The success of a community radio cannot be measured by audience ratings and revenues. What is important here is the effect that a broadcast series has on the targeted community. A radio show in a language spoken by a small minority will be highly appreciated by this group of people, although they would not be considered a big enough market to be targeted by other media.

There is a number of projects existing that aim to foster production of programmes for an international audience that can be aired all over Europe or even world wide. With AMARC's "Radio voix sans frontieres" (Radio without frontiers), more than 30 stations in Europe share a full day of live broadcasting once a year on the UN anti-racism day. Many exchange projects are oriented around themes of an international meaning, like migration, sustainable development or Human Rights.

What keeps programme exchange from becoming a day-to-day routine, is the technical and structural thresholds. The existing programme exchange platforms on the Internet are not connected to each other and do not follow a common metadata scheme and user interface, which makes it harder to search and offer content on an international scale, as every programme needs to be uploaded and indexed in different standards for every country. The user might not even be able to understand the language a sharing platform's website is written in.

Building on the experiences made at Orange 94.0 in Vienna and in other community radio stations in Austria, we researched the requirements of producers and editors for easy access to online archiving and content sharing. A common metadata scheme and the ability to search in all databases at once is a main criteria, the other is making access to the repositories as convenient as possible. Participating in exchange of audio content needs to work as easy as sending an email or using a search engine, with programme recording and uploading being done automatically.

Only an easy-to-use system that encourages more people to enter their day-to-day programmes into an international platform will create a critical mass of content that can be searched and re-used in local production. Such a platform would certainly help to strengthen the position of community radio and foster international cooperation in Europe.

3 The StreamOnTheFly Network

The project aims to create a network of radio content that develops in an organic and dynamic way, in decentralized structures that build synergies on an international,

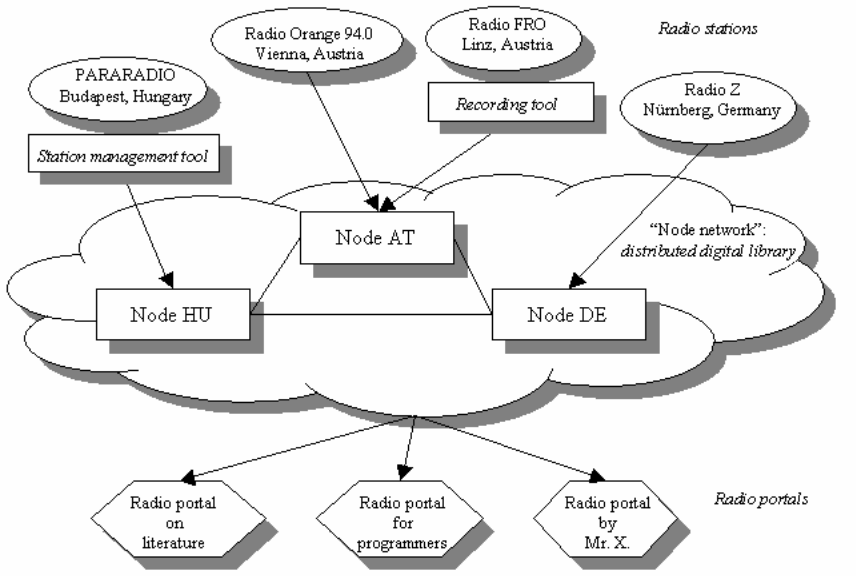


Fig. 1. Example for a StreamOnTheFly network

community-based level. The content is offered by local producers or stations to the network, but it's the international communities who choose which content is presented and how it is presented. International publishing on the net is not a form of archiving or recycling anymore, but a motivation to communicate with a whole new group of people.

The StreamOnTheFly network consists of the following elements: nodes, stations and portals (Fig. 1). Nodes together can be seen as a distributed digital library specialized for audio content. Stations send content to the nodes for archival, and portals present thematic selections of the contents of nodes.

For example, Orange 94.0, a community radio in Vienna, may send some of its broadcast material on the node running in Austria. These programmes are archived at the node and become accessible for the whole StreamOnTheFly network for a given period of time. A German radio may find a programme on the German node that they want to rebroadcast locally. They find the contact for this programme, in this case Radio Orange and ask for their permission for rebroadcast. If there is a portal in France for the Turkish people, its portal editor may include some of the Turkish programmes of Radio Orange in the recommendation list. Users may also go directly to a node, query/browse the whole content of the network, and listen to selected programmes.

3.1 Services of the Network

The StreamOnTheFly network tries to serve the whole lifecycle of a radio programme, starting with the creation and ending with various (re) uses and discussions of the programme.

Nodes. Nodes implement the basic digital library infrastructure for the network. Each node hosts a set of radio stations. For each station the node archives radio programmes with a rich set of metadata and other associated content. For example, the content may be present in several formats of audio, or photos may be attached to the audio content.

Users get transparent access to the contents of the whole network at any node. They can browse through radio stations or in topic trees or search in the archive. There is an advanced query facility, which enables registered users to construct and save complex queries to be executed frequently. Interesting items can be collected into a playlist. These playlists can also be seen as a kind of personalized radio. Users may listen or download audio content in different formats (e.g. MP3, Ogg/Vorbis) and in different bit rates or quality.

The network gathers and forwards information about radio programmes back to their creators. This involves the collection of various statistics (download, listen, etc.), the collection of comments, references and rating information.

Editors have a comfortable 'console screen' where they get an overview about all their programmes, they can publish new programmes and manage existing ones. Access privileges can be controlled on various objects: stations, series, programmes etc.

Stations. A station management tool has been developed by the project, which connects radio stations to the network. This tool provides an easy way to feed a node with radio programmes. Furthermore, it helps in the scheduling work of the station, and leads the creators' hands before and after the broadcast of a radio programme. Authors may add metadata to their programmes and publish them on a node. Programmes are sent to the node using XBMF format (see 3.2).

In the case when no prerecorded audio is available for a radio programme (e.g. live shows), a recording tool can be used. This software records the broadcasted audio of a station, and automatically cuts the stream into individual radio programmes.

Portals. In the StreamOnTheFly project a radio portal offers a selection of radio programmes in a custom design. Portals may serve ethnical groups, topics or geographical regions by selecting relevant content from the whole network. Additionally, a portal can offer editorials and more readings or pictures about the selected programmes. Visitors of a portal may listen to programmes individually or listen to a continuous stream playing the list of selected programmes over and over again. Visitors may also rate, comment and discuss programmes. Comments and ratings made on portals are forwarded to the nodes, and the comments are delivered to the creators of the programme via e-mail.

Editors can select programmes for the portal either one by one or using queries. In the first case editors assemble a playlist on the node and send it to the portal. In the second case a saved query from the node can be displayed in a box of the portal. The contents of such boxes will always be refreshed to show the current results of the query as executed on the node.

3.2 Issues on Metadata

At the time of the design phase the project team found many emerging activities for attaching metadata to multimedia. These solutions provide schemas for metadata and

methods to attach or embed metadata into the content (see SMPTE [1], AAF [2], MPEG-7 [3]). Schemas often contain an object hierarchy for metadata description and controlled vocabularies for the values of object properties. We found that these activities were overlapping and were all targeted at large radio and television stations and associations, with a highly detailed and complex format and structure. The project received feedback from various forums and radios that thought the use of these metadata formats is not practical for them and they would welcome a simpler and more suitable recommendation for metadata usage.

The European Broadcasting Union (EBU) prepared a recommendation on a core metadata set for radio archives [5] based on Dublin Core [4]. This document defines how to use the Dublin Core elements in the area of radio archives. StreamOnTheFly project participated in the launch of an initiative called Shared Online Media Archive (SOMA). The SOMA Group (consisting of AMARC, OneWorld, CMA and Panos) published the SOMA Metadata Element Set [6], which builds upon the EBU Core Metadata Set.

The SOMA metadata set became the part of the new exchange format called XBMF (Exchange Broadcast Binary and Metadata Format) defined by the project. XBMF is designed for transfer and archival of radio programmes coupled with metadata. This is a simple container format, in which metadata (in XML format), content files, and other associated files can be packaged together. XBMF may contain the main content in several audio formats, and any other files connected to the content (e.g. photos, documents, logos). XBMF can also be applied for video. The concept of XBMF was presented at community radios, and received a positive response.

StreamOnTheFly will use and support both the SOMA and the XBMF standards. Within the SOMA metadata standard, StreamOnTheFly will need to store some additional data. To assure compatibility there is an "additional data" field integrated into the SOMA standard by request of the StreamOnTheFly consortium.

3.3 The Operation of the Network

Unified access to the audio content was one of the main considerations in the network. In order to provide fast and robust search functionalities we decided to automatically replicate all metadata on each node of the network. As audio can take a large amount of disk space, the content is only available at the node where it has been published (its 'home node'). Each object in the network receives a globally unique identifier, which also provides information about the location and type of the object.

Transparent listening of audio content was solved with the help of an XML-RPC controlled streaming engine called Tamburine [8] and 'icecast', the well-known open source streaming server. When the user starts listening to a programme, the node asks Tamburine to handle the task of streaming. Tamburine then contacts the other Tamburine instance running on the home node of the programme, which starts to stream the audio to the user node. This transparent routing solution creates the possibility to provide a more reliable streaming solution with the use of caching and other optimization techniques.

Another requirement was to build the StreamOnTheFly network in a distributed, decentralized and self-organizing way. Most of the archives or digital libraries for audio and video in the world are not distributed. An example for a centralized service with similar goals to StreamOnTheFly is the Open Meta Archive

(http://oma.sourceforge.net/). Learning from distributed digital libraries (e.g. NCSTRL) and peer-to-peer file-sharing networks (e.g. Napster, Gnutella), StreamOnTheFly network goes further towards self-organization (described in more detail in section 4).

There is no central server in the network, each node has a set of neighbours, and nodes periodically exchange new and modified metadata with their neighbours. These neighbour connections draw the graph of the whole network. If a node is down, the rest of the network is still functional, and data flow can still find another way in the graph if there are enough 'redundant' connections. Multiple neighbour connections also help to speed up metadata propagation and correct data loss or errors in replication. On the other hand, nodes with a low budget can lower their network traffic by decreasing frequency of metadata exchange.

Nodes accept data exchange requests only from allowed neighbours, a list maintained by the node administrators. In this way authorization is decentralized as well: if a node wants to be the part of a network, it needs to establish a neighbour relationship with at least one node from that network.

The network uses a few controlled vocabularies for descriptive metadata, e.g. topic or genre names. Users at any node with proper privileges can translate entries into local languages or add new entries to vocabularies. The propagation of the changes is done with the same mechanism as metadata replication.

The framework was implemented using PHP and Postgres DBMS. All user interfaces are accessible via web browsers. HTTP and XML-RPC are used for communication between components. The framework was built using open source and free software components and tools, and the framework itself will also be made available to the public.

4 Principles of Self-organization

StreamOnTheFly network architecture is based on the principles of self-organization. Self-organization in community life (such as within the community of radios stations in Europe) means that the (social) rules governing the activities within the community evolve in time in order to adapt the community to the new challenges appearing within this sector. In system theory the field of self-organization seeks general rules about the growth and evolution of systemic structure, the forms it might take, and finally methods that predict the future organization that will result from changes made to the underlying components. [9]

Distributed data repositories based on the principles of self-organization should reflect the community (social) structure and it is important that repositories could be adapted according to their evolutions. Key features for self-organizing networks are the following: this type of networks encourages users to contribute information. The StreamOnTheFly network helps editors to contribute radio shows. The network can thus be considered as a virtual platform for collaborative, community-based radio content production, distribution, and evaluation. Of equal importance, self-organizing networks aggregate and qualify the most relevant information in (semi-) automated fashion. The results are twofold: users of the systems can find quality information with confidence, while administrators and developers require less work to manually sift through large amounts of information. StreamOnTheFly network provides

services for content rating (and other statistical information about the actual use of audio content) and backward propagation of this information to the authors. Self-organizing networks create a community-like environment, and attempt to emphasize and visualize relationships between users thus increase user participation [10]. StreamOnTheFly network users are clustered according to their roles (e.g. station manager, editor, series editor, metadata classification scheme editor, general listener, etc.) and system is structured according to their feature requirements. The lack of central control within StreamOnTheFly network reflects the actual rather chaotic community structure of European community radios, and the easy way of joining and leaving the network mimics the dynamics of movements in this sector.

The StreamOnTheFly system is a special example that is also based on the general principles of computational self-reflection [11] but the self-reflected system architecture (in which the self description of the actual system is an active part of the system itself) is substituted by a live human community and its social structure. An active (social) community (the community radios in Europe) is thus in the loop, the system is structured according to their (social) relationships. The community mimed system (service) structure emphasize the openness, thus the system is rather an open infrastructure than a closed and static network, in which users can change and adapt the system itself to their own patterns of usage. Adaptation is possible on both syntactic and semantic level via the portal and the metadata related service components. Dynamics within the community can also be reflected easily. New StreamOnTheFly nodes can join or leave according to the actual changes in the community. A new classification scheme or evolution of the actual classification scheme can immediately be embedded in an automatic way.

Summarizing the needs for a self-organizing collaborative media-sharing network, the following principles were used:

- No central role, server or governance, rather provision of an open, evolving infrastructure
- Easy to join and leave, but keep the possibility to limit or exclude partners
- Free and unified access and content publishing
- Support for personalization on several levels (e.g. content, user interface, community)
- Support for information (e.g. awareness, rating) propagation also back to the authors

5 Conclusions and future work

The project is after the second validation scenario. In this scenario everyday work in a radio station was simulated with the software. These test sessions justified the requirement of a station management tool, in order to minimize the effort needed to publish a programme on the network. Using this tool can also be an incentive for radio stations operating with home-brewn station management solutions to join the network. Several radio stations are waiting to become part of the network, and some existing networks with simpler infrastructure are also considering joining in. In order to connect the network into a wider but simpler level of distribution, an Open Archive Initiative compliant gateway will be added to StreamOnTheFly.

With StreamOnTheFly being fully implemented and tested, we will start to set up a number of nodes all over Europe, together with partners and regional umbrella organizations. Existing platforms will be encouraged to participate in the network, by changing to the StreamOnTheFly system or by creating interfaces to it. Several series and broadcasts, especially those for an international target audience, will be uploaded and presented in showcases on our own radio portals, to show the potentials of StreamOnTheFly and encourage more editors to participate.

6 Acknowledgements

Authors would like to thank to the participants of the project for support and help in the design and development of StreamOnTheFly software: Roland Alton-Scheidl, Thomas Hassan and Alexey Kulikov from Public Voice Lab, Jaromil, an independent audio/streaming expert, Thomas Thurner, Wolfgang Fuchs and Roland Jankowski from Team Teichenberg, Tamas Kezdi, Mate Pataki and Gergo Kiss from MTA SZTAKI DSD.

This work is supported by the EU under contract IST-2001-32226.

References

1. Society of Motion Picture and Television Engineers (SMPTE), http://www.smpte.org
2. AAF (Advanced Authoring Format) Association, http://www.aafassociation.org/
3. MPEG-7, http://www.mpeg-industry.com/
4. Dublin Core Metadata Initiative, http://dublincore.org/
5. EBU Core Metadata Set for Radio Archives (Tech3293), http://www.ebu.ch/tech_32/tech_t3293.pdf
6. Shared Online Media Archive (SOMA) metadata format 1.0 http://soma-dev.sourceforge.net/
7. StreamOnTheFly project homepage, http://www.streamonthefly.org
8. Tamburine: http://tamburine.dyne.org
9. Self-Organizing Systems (SOS), http://www.calresco.org/sos/sosfaq.htm
10. Hisham Alam: Self-Organizing Sites. http://www.newarchitectmag.com/archives/2002/01/alam/
11. Paul Dourish: Developing A Reflective Model of Collaborative Systems. ACM Transactions on Computer-Human Interaction, Vol. 2, No. 1, pages 40-63, 1995

A P2P and SOA Infrastructure for Distributed Ontology-Based Knowledge Management

Nektarios Gioldasis, Nikos Pappas, Fotis Kazasis, George Anestis, Stavros Christodoulakis

Lab. Of Distributed Multimedia Information Systems
Technical University of Crete (MUSIC/TUC)
University Campus, Kounoupidiana, Chania, Greece
[nektarios,nikos,fotis,ganest,stavros]@ced.tuc.gr

Abstract. The Digital Business Ecosystem Integrated Project funded by the EU aims at the development of a Europe-Wide virtual economic environment where European SMEs will advertise themselves and their service offerings with the purpose of competing, collaborating and co-evolving with other SMEs. One of the main strategic goals for this project is to model, design and implement such an environment based on observations made on biological ecosystems where several species live together, compete and co-evolve as the physical environment of the ecosystem changes. From a technical point of view, the DBE will provide a Distributed Open-Source Infrastructure which will be the glue that will bring together the European SMEs and will enable knowledge sharing and intelligent discovery of partners and services. The focus of this paper is on the Knowledge Management Infrastructure of the DBE. We firstly discuss the problem, we then present the proposed approach, and finally we identify some of the research issues that we will concentrate on.

1 Introduction

J.F. Moore describes a Business Ecosystem as “An economic community supported by a foundation of interacting organizations and individuals - the organisms of the business world. This economic community produces goods and services of value to customers, who are themselves members of the ecosystem. Over time, they co-evolve their capabilities and roles, and tend to align themselves with the future directions...” [1]. The Digital Business Ecosystem is the enabling technology for the Business Ecosystem. A Digital Business Ecosystem is defined as "evolutionary self-organising system aimed at creating a digital software environment for small organisations" that support regional and local development by empowering open, distributed and adaptive technologies and evolutionary business models for the growth of small organisations.

The Digital Business Ecosystem vision is in fact based on two fundamental ideas: those of self-organisation and biological evolution. The concept of self-organisation implies intelligent behaviour and the ability to learn on a short time scale, whereas

evolution implies an ability of the system to optimise itself through differentiation and selection of its components on a long time scale. These ideas can be only realized if the (eco) system as a whole is able to learn over time and adapt to this knowledge that itself produces as well as to knowledge that it is derived from its biosphere.

Thus, a fundamental objective of DBE is that it aims to build an infrastructure for supporting Knowledge-Based Business Communities. The Knowledge Base of such a system is one of its core infrastructural components and it will be the enabling factor for the advanced features that were previously described. Within the DBE the purpose of Knowledge Base is to provide a common and consistent description of the DBE world and its dynamics, as well as the external factors of the biosphere affecting it. The Knowledge Base will provide a consistent Knowledge Representation Model for storing Business and Service Ontologies, Business and Service Descriptions, Regulatory Framework, and usage history regarding the day-to-day activities in the DBE. This knowledge will be exploited by other core components for many purposes. One of the main usages of the Knowledge Base is for enabling valuable recommendations (in terms of partnerships and evolutionary actions) by the recommendation infrastructural service to the SMEs "living" in the ecosystem. Personalized recommendations are provided by a special system component, the recommender, which takes into account business and service models, usage histories, and fitness parameters, in order to recommend best-suited partners and services for the user SMEs. The knowledge Base will store the suggestions made by the recommender, in order to provide feedback for the evolutionary procedures, which will lead to the long-term study and adaptation.

This paper presents ongoing work done in the DBE project. Its focus is on the knowledge modelling and management requirements and the architectural approach envisioned for DBE Knowledge Base Infrastructure.

2 The DBE Knowledge Base Requirements

The DBE Knowledge Base (KB) provides a common and consistent description of the DBE world and its dynamics, as well as the external factors of the biosphere affecting it.

Its content includes, among others, representations of the SME's Business Models and Ontologies, the Service Models and Ontologies, the SME views of the ecosystem, the user models, and models for gathering statistics. The KB is being used in order to provide a consistent knowledge model and input for the Service Description / Business Modeling Language, the recommendation process and the Service Composition process.

An important aspect of the KB management is to support the sharing of its content. For that it should follow a scalable design/implementation that can support virtually any number of DBE users as well as organization-wide transactions and/or cooperation. The KB provides a variety of advanced content services (and tools) that include the following:

- **Content management services:**
  these services provide functionality for storing, finding, retrieving and presenting any type(s) of the aforementioned content that it hosts.
- **Content personalization services:**
  the services of this type utilize DBE user profiles and content access patterns to create and/or present customized content to each DBE user.
- **Content group services:**
  such services provide scheduled delivery of content according to predefined business rules, as well as packaging the content for individual DBE users (e.g. services for DBE users that participate to a special cooperation schema).
- **Content aggregation services:**
  These services include automatic combination of content from a variety of content sources and (probably) formats (i.e. other KBs and/or legacy systems).

Another important aspect of the KB management is to support the sharing of its content in a P2P environment. At first glance, many of the challenges in designing P2P systems seem to fall clearly under the banner of the distributed systems community. However, upon closer examination, the fundamental problem in most P2P systems is the placement and retrieval of data. Indeed, current P2P systems focus strictly on handling semantics-free, large-granularity requests for objects by identifier (typically a name), which both limits their utility and restricts the techniques that might be employed to distribute the data. These current content sharing systems are largely limited to applications in which objects are large, opaque, and atomic, and whose content is well-described by their name; for instance, today's P2P systems do not emphasize content-based retrieval of text files or fetching only the abstracts from a set of text documents. Moreover, they are limited to caching, pre-fetching, or pushing of content at the object level, and know nothing of overlap between objects.

These limitations arise because the current P2P world is lacking in the areas of semantics, data transformation, and data relationships, yet these are some of the core strengths of the data management community. Organizational Information is complex and has complex relationships in the real business world, and the DBE seeks to capture and manage such information. Queries, views, and integrity constraints can be used to express relationships between existing objects and to define new objects in terms of old ones. Complex queries can be posed across multiple sources, and the results of one query can be materialized and used to answer other queries. Data management techniques can be used to develop better solutions to the data placement problem at the heart of any P2P system design: data must be placed in strategic locations and then used to improve query performance. On the other hand businesses in the real world are independent and they may cooperate, but at the same time compete. The environment is clearly a P2P environment where some, but not all the information of the business is revealed to the outside world. From the above, it becomes clear that the use of relational database management technology for managing the DBE KB content is very important , and it can leverage the increased scalability, reliability, and performance of a successful P2P architecture for the DBE project.

The DBE KB supports the Recommendation Services that will act as an autonomous processes that manage SME preferences (either business preferences or service preferences) and matches theses preferences with available business

descriptions and service descriptions. An important assumption and investigation focus of DBE which is used in the design and implementation of the Recommendation mechanisms is the existence of powerful business and service Ontologies that capture the semantics of business models and service descriptions. These Ontologies are used to define the corresponding preferences for businesses and services. These preferences are attached to the business model of each SME in the DBE environment. The Ontologies within DBE are described in an XML based language (XMI) and mapping mechanisms will be used to ensure the transformation from different Ontology languages to the standard Ontology descriptions used in the DBE. The DBE Ontologies can thus be used by the recommendation mechanisms regardless of the particular models used by each SME in the DBE environment.

3 A Preliminary Knowledge Management Architecture

In this section we present a preleminary conceptual architecture (i.e. a set of logical layers and components and relationships) for the DBE Knowledge Base Infrastructure and other system components that directly or indirectly exploit this infrastructure. The purpose of this architecture is to show how the DBE Knowledge Base will be perceived by the rest DBE System Components and how it will be exploited by them.

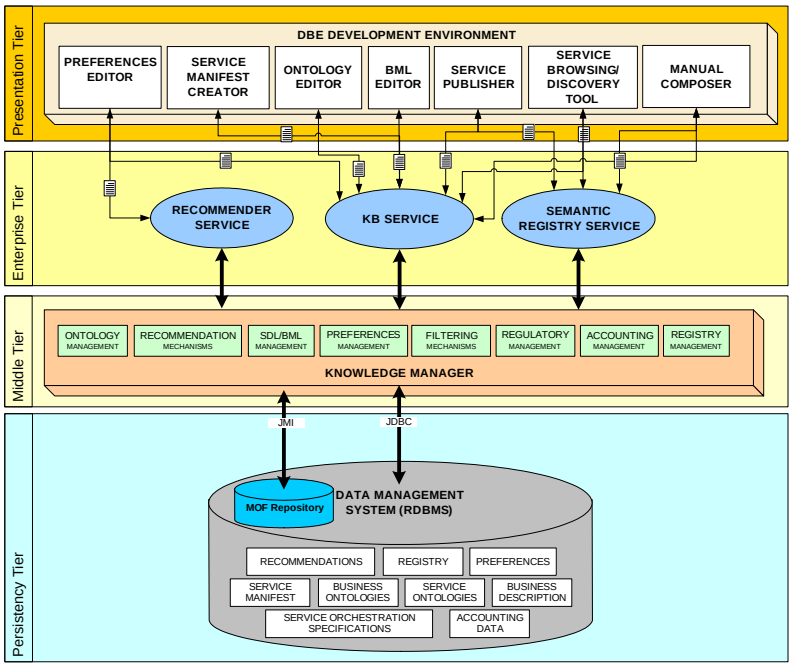


Figure 1 : Conceptual Architecture of the DBE KB Infrastructure

The abstract architecture of the DBE Knowledge Base illustrated in Figure 1 clearly distinguishes between the client modules (front-end applications that provide

GUIs with which one can create and access knowledge), the enterprise-level services (a set of core services that expose the functionality of the KB to the client modules), the middle-tier management modules (which implement the core logic of the Knowledge Base) and the back-end persistency infrastructure that permanently stores the DBE Knowledge.

3.1 System Components

In this section we briefly describe the main components that appear in Figure 1 and in particular their use with respect to Knowledge Base Infrastructure. We start by describing the Core DBE Services that are used to export the functionality of the DBE Knowledge Base Infrastructure to the rest System and then we describe several components that exploit the aforementioned services and are used to create, store, modify, and present Knowledge which is finally stored in the DBE Knowledge Base.

From a technical point of view, the DBE KB will implement a MOF-Based Repository utilizing the Java Metadata Interface (JMI) [15] specification. This approach will provide a standard and flexible mechanism for handling different knowledge models. The persistency of the KB contents will be achieved through a java-based relational database management system providing this way a fully platform independent solution. Thus, meta-data browsing can be done through standard interfaces (JMI) while advanced recommendation mechanisms (requiring powerful querying features) can be implemented on-top of the RDBMS where the real contents are stored.

- KB SERVICE:The KB Service is a Core DBE Service that encapsulates all the functionality of the DBE Knowledge Base and provides a standard interface through which the other DBE components can use this functionality in a coherent and consistent manner. Each higher level application that wants to exploit the Knowledge Base has to invoke the required KB Service operations. The core logic of the KB service is implemented by the Knowledge Base Manager which is responsible for semantically organizing and managing the various KB Data as well as handling the requests for accessing those data that are coming through the KB Service. In the distributed DBE environment, although the KB Service will be perceived as a single service, it will actually be a set of KB Service Instances (each instance will be hosted in an SME node) that will be self-coordinated and will operate as a whole over a P2P network following this way a SOA approach.
- RECOMMENDER SERVICE:The Recommender Service is a Core DBE Service responsible of handling SME preferences that specify the needs of each independent SME in the DBE environment in terms of partnerships with other SMEs as well as services needed to compose more complex services for its customers. It provides a standard interface that hides the technical details of the underlying Recommendation Mechanisms that implement powerful matching algorithms using the DBE Business and Service Ontologies. The produced recommendations are permanently stored in the Data Management System (Persistency Tier). As with the KB Service, in the distributed DBE environment, this service will be a set of Recommender Service Instances over a P2P network.

- SEMANTIC REGISTRY SERVICE: It provides a standard interface capable to implement a Semantic Registry Service functionality required in the DBE environment. The Semantic Registry Service provides standard representation hierarchies and query facilities provided by the standard registries. It should be noted that this service simulates the functionality of a Semantic Registry and it is not a repository itself. On the contrary, it exploits the Data Management System (Persistency Tier) for the storage and retrieval of its contents. At the Middle-Tier a Registry Manager is used to implement the advanced functionality of this Service and to handle the requests for retrieving and accessing the stored content. The Semantic Registry Service is a Core DBE service and from a technical point of view is a set of Semantic Registry Service instances that are hosted on special peers (super peers) over a P2P network.
- ONTOLOGY EDITOR: The Ontology Editor is a component that provides a GUI to the end user and it is used for the management (creation/update/retrieve) of the Ontologies (Service and Business Ontologies) that are developed and used in the DBE environment. In order to import a new ontology in the DBE an extension of the DBE Knowledge Base is required. Such a task is under the responsibility of the Ontology Editor. It accesses the KB by "talking" to the (local instance of the1) KB Service for manipulating the various Ontologies (store/retrieve/update Ontologies). The mechanisms that perform the real manipulation of the Ontologies are implemented at the Middle-Tier by the Ontology Management Module. The final modifications are stored in the persistent Data Management System.
- BML EDITOR: It is a Tool that resides on each DBE node (at least on the Nodes that provide DBE Services) and it uses the Business Ontologies (created with the Ontology Editor) in order to describe SMEs according to their business models, policies, assets, competencies, partners, etc. It also uses the Service Ontologies in order to describe the services that are offered by the SMEs. It interacts with the KB Service for accessing Business Ontologies and storing/updating/retrieving Business Descriptions and Service Descriptions.
- SERVICE MANIFEST CREATOR: It is a Tool that resides on each DBE node (at least on the Nodes that provide DBE Services) and it is used to integrate the BML Description and SDL Description of DBE Services producing their Service Manifest. It interacts with the local instance of the KB Service to store/update/retrieve Service Manifests.
- SERVICE PUBLISHER: It is a tool that is used to publish a Service Manifest to the Semantic Registry. It "talks" with the KB Service in order to retrieve Service Manifests and with the Semantic Registry Service in order to publish/update/remove Service Manifests (the published ones). The Semantic Registry Service is based on the Data Management System for the storage of this content.
- SERVICE BROWSING/DISCOVERY TOOL: It is a Tool that resides on each SME node and it is used to contact the Semantic Registry Service through its local instance in order to browse and retrieve the contents of the Semantic Registry of the DBE. This tool has two modes: The simple one, where it just browses the

[1] Local Instance means that both the KB Service Client (e.g. the Ontology Editor) and the Service Instance are running on the same host.

registry for Services, and the advanced one where using Business and Service Ontologies it provides guidance for query formulation when a user looks for particular services. In this mode, the discovery of services is actually an iterative process through which the user is guided to form the query request that expresses his/her needs in the best possible way. After the query formulation the user submits the query and gets a ranked set of results (Published Service Manifests). Beyond the interaction with the Semantic Registry Service, this tool needs to interact with the KB Service in order to retrieve (parts of) Business and Service Ontologies for guiding the user in formulating queries.

- PREFERENCES EDITOR: Using Service and Business Ontologies an SME describes its preferences for desired services. SMEs are also able to describe business requirements that must be satisfied by the providers of those services (e.g. business policies, accounting policies, etc.). The SME may ask explicitly for recommendations according to its preferences or may browse the recommendations that the Recommender Service had automatically produced. The Preferences Editor is a tool that provides a GUI and uses the KB Service for storing/retrieving and updating preferences, and accessing Ontologies. It also uses the Recommender Service in order to ask for explicit recommendations according to the defined preferences. At the middle-tier special management modules implement the advanced functionality that is required for the realization of the aforementioned behavior.
- MANUAL COMPOSER: The Manual Composer component provides a GUI through which the end-user can compose new services. It interacts with the Semantic Registry Service or explores the suggestions made by the Recommender Service in order to discover candidate constituent services and to publish the newly created (composite) services. Also it interacts with the KB Service in order to store/retrieve/update Orchestration Specifications and Service Manifests of Composite Services and to retrieve the Required Ontologies for performing the required Service Matchmaking.

4 Knowledge Management in P2P Environment

In the DBE environment a large number of SME nodes will be pooled together to share their resources, information and services while keeping themselves fully autonomous. The SMEs will share data that will have rich structure and semantics, and thus advanced mechanisms for querying and discovering knowledge from these data are needed.

Taking these into account above, the DBE knowledge base should be built upon an efficient P2P data management system. The tasks during the development of the DBE KB include conceptual modeling, exhaustive study of current P2P data sharing technologies and investigation of strategies for deploying data management systems over P2P networks.

The design of the DBE KB should take into account the benefits of the P2P networks that can be concluded in adaptation, self-organization, load balancing, fault

tolerance and high availability. Despite these benefits the deployment of P2P networks can introduce in many cases performance and consistency problems. Also current P2P solutions do not support advanced mechanisms for data management.

The challenge in designing and developing the DBE KB is to enable the effective support of rich semantics, data transformation, data relationships, data constrains and complex queries across multiple sources in the DBE network. KB will have to place data in strategic locations in order to improve the performance of the retrieval mechanisms, since the data sharing of enormous amounts of data is useless without advanced search mechanisms that will allow SMEs to quickly locate and use the desired information.

Current service oriented architectures and standards almost completely ignore the modeling of the business environment including business models, business processes, business environment models, domain specific specializations, instantiations etc. In addition many of the service oriented standards proposed are too close to a centralized or "one-owner" environment, and not a real P2P environment. The objective of the DBE KB is to organize, store, and efficiently retrieve description metadata about businesses and services. The KB will have to handle a lot of metadata that follow different schemas, since a great variety of business and service models will exist. Thus, techniques and strategies for message and query routing in schema-based P2P systems (should be investigated and deployed([8],[9],[10] The KB is responsible for matching and accessing semantic data across the DBE network nodes by offering to the SME users a set of Core Services for transparently searching and accessing DBE Knowledge.

The search mechanism to be deployed determines the behavior of peers in three areas: Topology (how peers are connected to each other), Data placement (how data or metadata are distributed across the network of peers) and Message routing (how messages are propagated through the network) [2].

While designing the search mechanism and defining the peer behavior, the query language, the returned results and the autonomy of peers (e.g. an SME may wish to define what data wants to store, or share data only within a group of SMEs) should also be considered.

A P2P model can either be pure or it can be hybrid. In a pure model, there is no centralized server (e.g. Gnutella and Freenet). In a hybrid model, a server is approached first to obtain meta-information, such as the identity of the peer on which some information is stored, or to verify security credentials. From then on, the P2P communication is performed. Examples of a hybrid model include Napster, Groove, Aimster. There also intermediate solutions with Super Peers, such as KaZaa. Super Peers contain some of the information that others may not have. Other peers typically lookup information at Super Peers if they cannot find it otherwise.

Pure P2P systems tend to flood the network with query messages and the limited capabilities of some peers cause performance bottlenecks. So, pure P2P systems present efficiency weaknesses that make them unsuitable for the DBE. On the other hand hybrid systems perform better but they are not scalable and fault tolerant.

Super peer systems are trying to combine the two previous approaches to take advantage of the efficiency of a centralized search and the autonomy, load balancing and robustness of a distributed search. Super-peer based P2P infrastructures usually exploit a two-phase routing architecture, which queries first in the super-peer

backbone, and then distributes them to the peers connected to the super peers. Super-peer routing is based on different kinds of indexing and routing tables [6], [7]

The DBE KB architecture follows the super peer network paradigm for the efficiency benefits that it presents and its capability of taking advantage of the heterogeneity of the peers by assigning greater responsibility to those peers that are more capable to handle it [3]. However the choice of the super peer model does not solve all the problems that the DBE KB will face, since the design should consider several of challenging issues like: dynamic self-organization of peers and super peers, performance trade offs, load-balancing among equivalent peers and among simple peers and super peers, avoidance of single-point of failure in the super peers, search performance using super peers, data placement and indexing across super peers and other research issues.

In the DBE environment many IT systems will have to be integrated in a common network. These systems will be developed, managed and integrated using a range of methodologies, tools and middleware. We have witnessed during the last few years, especially as a result of efforts at OMG and W3C a gradual move to more complete semantic models as well as data representation interchange standards. OMG's Model Drivel Architecture (MDA) provides an open, vendor-neutral approach to the challenge of interoperability, building upon and leveraging the value of OMG's established modeling standards: Unified Modeling Language (UML), Meta-Object Facility (MOF), XML Metadata Interchange XMI etc. [12,13,14].

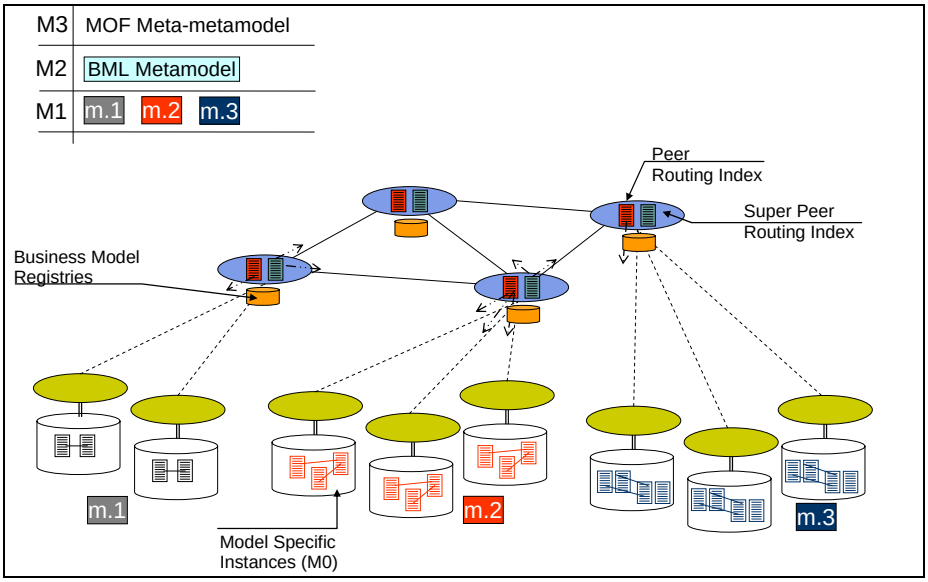


Figure 2: The DBE KB P2P Infrastructure

One of the most challenging objectives for the DBE is to enable common business understanding and at the same time to provide for differentiation between businesses and organisations. From a technical point of view, this means that while each SME may have its own model for service or business description, at the same time it needs

to communicate with other SMEs and to understand each other. We have decided to follow a multi-layered approach in the Knowledge Modelling and Representation where each level in a top-down fashion adds more semantics into knowledge modelling concepts through specialization. For that, we are adhering to the rules of the MDA technology [12]. The top layer (M3) in MDA is the MOF meta-meta-model [13] that provides a mechanism for defining meta-models (meta-meta-data) which are actually residing at the M2 layer. M2 meta-models are used to provide specific models (meta-data) of representing knowledge (data). Thus the various types of content included in the KB are represented via XMI [14] so that multiple participants can query the stored content in a standardized and consistent manner. The KB Manager is responsible for the implementation of the appropriate structures for managing contents described with XMI. The manager uses the metamodel descriptions to determine how to manage contents that are actually instances of these meta-models. Figure 2 shows the DBE approach to metadata layering.

The P2P Knowledge Base infrastructure is shown in Figure 3. Three instances of core DBE services run at each SME node that belongs to the DBE network: the Recommender Service (RS), the Semantic Registry Service (SRS), and the KB Service (KBS). These services cooperate with the other instances in the peer network to perform the distributed knowledge update or retrieval tasks. These core services export a transparent common interface to DBE applications that need access in the DBE KB and hide the complexity of the necessary P2P communications between the service instances.

As we have described above, the distributed KB will follow the super peer network paradigm. Some SME peers that have enough bandwidth and processing capabilities will have more responsibilities than other SME peers. These SMEs will also run the three core service instances (of the KB infrastructure) that will behave differently than the service instances of the simple SME peers. The SME super peers will form a network among them and will be aware of the topology. These super peers act as servers to a group of SMEs are accepting recommendation, knowledge extraction and service discovery requests from their clients and based on the information distribution strategy that will be applied, forward the requests to the appropriate super peers that are able to satisfy or can reference to simple nodes that can satisfy the submitted requests.

For example the Service Publisher application (as shown in Figure 3) publishes the Service Manifest Advertisement (probably a summary of the Service Manifest that contains the most valuable information of a Service Description) accessing the local instance of the Semantic Registry Service. The local instance communicates with the respective service instance of the super peer that the SME is connected to and sends the advertisement. The super peer Service processes the Service Manifest Advertisement and based on the distribution strategy either it stores the service description in the local registry node (data management system) or it forwards the advertisement to the appropriate super peer Semantic Registry Service where it should be stored. The KB service acts in the same way as the Semantic Registry service. For example, BML descriptions produced by the BML Editor are pushed to the local KB-Service. This Service processes the description and forwards the public description to the super peer KB-service.

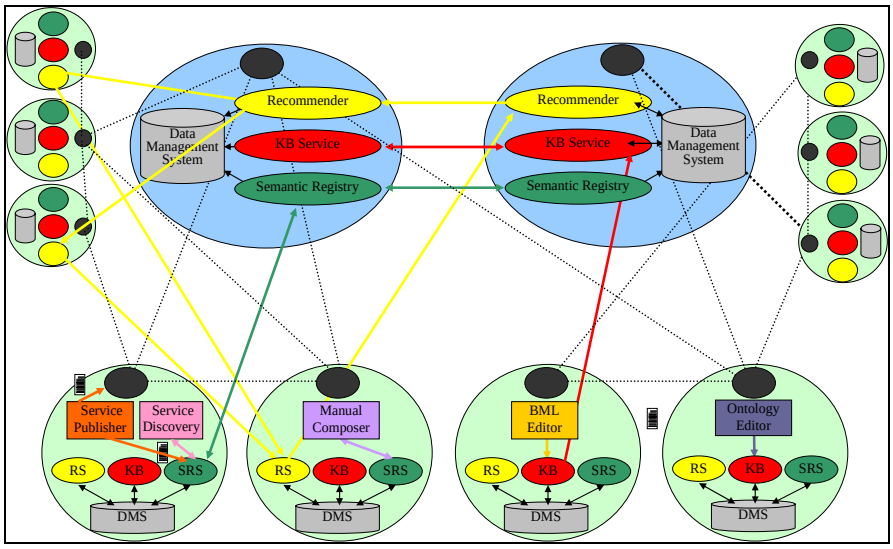


Figure 3: The Core Service Components of the DBE KB Infrastructure

The super peer KB service identifies the appropriate super peer that will handle the storage of the public information to the data management systems contributed by other SME peers. On the other hand the Recommender service acts autonomously. Periodically the local Recommender Service Instance asks the super peer Recommender Service for new system recommendations. The super peer Recommender service based on declared or automatically produced SME preferences in cooperation with other super peers identifies a group of SMEs that might contain information or services that should be recommended to the SME. The recommendation request is forwarded then to the Recommender Service instances of the qualified SMEs where is being processed and the recommendation results are sent (in P2P manner) back to the SME Recommender Service that requested them.

5 Conclusions and Open Issues

The previously described infrastructure for knowledge management will be an ontology-based P2P meta-data management system that its architecture can be used in many different environments. Some of the research issues related to the P2P Knowledge Management that will be examined during the project are the following:

- Ontology management (insertion, maintenance, conflict resolution and utilization) in P2P systems with the use of Relational Databases at each peer following a Service Oriented Architecture.
- Business model ontologies, business process ontologies, environmental ontologies, domain specific ontologies and their use and interplay in a dynamic service environment.
- Distributed Semantic Recommendation and Service Composition mechanisms

- Self-organization of the P2P network.

6 Acknowledgements

This work is being carried out in the scope of the DBE Integrated Project (IP:507953) funded by EU under FP6.

References

1. J.F Moore, The Death of Competition, 1996, pag.6-7
2. Neil Daswani, Hector Garcia-Molina, Beverly Yang "Open Problems in Data-Sharing Peer-to-Peer Systems" International Conference on Database Theory, ICDT 2003
3. Beverly Yang, Hector Garcia-Molina "Designing a Super-Peer Network" IEEE International Conference on Data Engineering 2003
4. Steven Gribble, Alon Halevy, Zachary Ives, Maya Rodrig, Dan Suciu. "What Can Databases Do for Peer-to-Peer?" WebDB Workshop on Databases and the Web, 2001.
5. W. Nejdl, W. Siberski, M. Sintek" Design Issues and Challenges for RDF- and Schema-Based Peer-to-Peer Systems" Technical Report, July 2003
6. B.Yang, H. Garcia-Molina "Improving search in peer-to-peer systems". Proceedings of the 22nd International Conference on Distributed Computing Systems 2002
7. Crespo, H. Garcia-Molina "Routing indices for peer-to-peer systems". Proceedings of the International Conference on Distributed Computing Systems 2002
8. W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson. M. Palmer, T. Risch "EDUTELLA: A P2P Networking Infrastructure Based on RDF". Proceedings of the International World Wide Web Conference 2002
9. W. Nejdl, M.Wolpers, W.Siberski, C. Schmitz, M. Schlosser, I. Brunkhorst, A. Loser "Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks. Proceedings of the International World Wide Web Conference 2003
10. M.Schlosser, M.Sintek, S. Decker, W. Nejdl "HyperCup – Hypercubes, Ontologies and Efficient Search on P2P Networks. International Workshop on Agents and Peer-to-Peer Computing 2002
11. OWL Services Coalition: OWL-S:Semantic Mark-Up for Web Services", http://www.daml.org/services/owl-s/1.0/owl-s.pdf
12. Object Management Group: "Model Driven Architecture a Technical Perspective", http://www.omg.org/cgi-bin/apps/doc?ormsc/01-07-01.pdf
13. Object Management Group: "Meta Object Facility Specification", http://www.omg.org/cgibin/apps/doc?formal/02-04-03.pdf
14. Object Management Group: "XML Metadata Interchange Specification", http://www.omg.org/technology/documents/formal/xmi.htm
15. Java Community Process "Java Metadata Interface Specification", http://www.jcp.org/aboutJava/communityprocess/review/jsr040/"

A Hierarchical Super Peer Network for Distributed Artifacts

Ludger Bischofs[1], Wilhelm Hasselbring[1], Jürgen Schlegelmilch[2], and Ulrike Steffens[2]

[1] University of Oldenburg, FK-2, Software Engineering Group, PO Box 2503, 26111 Oldenburg, Germany
{ludger.bischofs|hasselbring}@informatik.uni-oldenburg.de
[2] OFFIS, Escherweg 2, 26121 Oldenburg, Germany
{ulrike.steffens|juergen.schlegelmilch}@offis.de

Abstract. The transition from traditional paper libraries to digital libraries enables new strategies for the use and maintenance of artifact collections. Distributed software development can be regarded as a special case of digital library utilization, where developers or groups of developers are working on the same software geographically dispersed in time zones which might differ. We present a hierarchical super peer network which represents the organizational structures of distributed software development in a natural way and is able to integrate distributed resources like version control systems as well as local devices. This approach is then generalized to support the self-organization of widely distributed, loosely coupled, and autonomous digital library systems.

1 Introduction

The transition from traditional paper libraries to digital libraries enables new strategies for the use and maintenance of artifact collections. Collections are globally distributed and maintained by different organizations and even private persons. Digital binding techniques allow for the construction of project specific reference libraries by reorganizing existing library material and for the reintegration of project results [1]. As production, storage and classification of documents are now accomplished digitally, library support for intermediate and final results of collaborative writing can be achieved. Furthermore the collection and organization of assets other than documents, as for example works of art or services, is possible by referencing them from within the digital library.

Taking advantage of digital libraries in the described manner calls for a flexible support by a system architecture which enables the combination of collections against the background of different organizational, topical, and technical contexts, offering simple access to potential library users on the one hand and guaranteeing autonomy to library patrons on the other hand.

Distributed software development can be regarded as a special case of digital library utilization, where developers or groups of developers are working on the same software geographically dispersed in time zones which might differ. As the

use of a central repository for shared artifacts might have substantial drawbacks like slower and less reliable network connections, software developers rely on distributed artifact collections, or in other words on distributed digital libraries of software engineering artifacts.

In this paper we introduce a hierarchical super peer network for software development as an example for a flexible distributed digital library architecture. Distributed teams, in particular for open source software projects, can be regarded as peer-to-peer systems themselves. The support granted to them by the super peer network begins with the formation of new developer groups and projects and enables the flexible self-organization of the involved organizational units and their respective relationships. Apart from artifacts, distributed resources like version control systems as well as local devices can be integrated, so that developers are able to access and use shared artifacts any time and anywhere within the network.

Beyond the context of software engineering, hierarchical super peer networks can be regarded as a general approach to achieve flexibility and autonomy for a loose coupling of digital library collections. For this purpose, the organizational units and their relationships in the area of software engineering have to be reconsidered and transformed to more general units applicable to digital libraries in general.

This paper is organized as follows. Section 2 describes peer types representing the four central organizational units in the area of software engineering and maps the structures of distributed software development projects into a peer-to-peer architecture. The integration of additional resources like version control systems is demonstrated in section 3. Section 4 shows how peers can be organized in a hierarchical peer-to-peer network and how an appropriate lookup service can be designed. Section 5 generalizes the presented approach and describes a super peer network to support cooperation between arbitrary distributed digital libraries and collections. After presenting related work in section 6 we conclude and outline some future work in section 7.

2 Organizational Units and their Relationships

Peer-to-peer architectures are often characterized as the opposite of Client/Server architectures. The most distinctive difference is that in peer-to-peer networks the peers are capable of acting as client and server at the same time. Furthermore, peers are accessible by other peers directly without passing intermediary entities [2].

In case of distributed software development each developer can be considered a peer. A *developer peer* can offer and access artifacts within the peer-to-peer network. Developers are often organized in groups which are managed by special *group peers*. Beyond, developers and groups of developers are organized in projects to reach a common goal. A *project peer* offers the needed project management services. Organizations (e.g. an enterprise or institution) consist of projects, groups and developers and are managed by *organization peers*. The en-

tirety of the described organizational units constitutes a hierarchical structure. Figure 1 depicts a logical view of a possible structure of peers which does not necessarily reflect the physical structure of the involved computers. P_2 and G_3 for example could physically reside on the same computer.

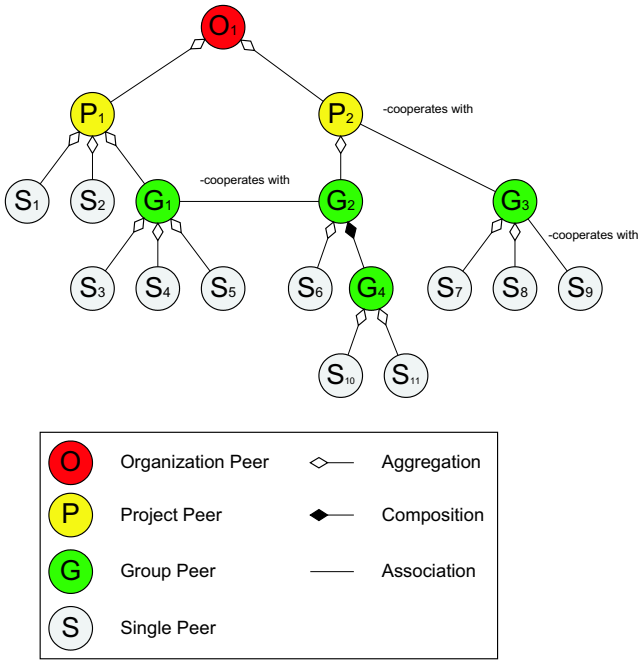


Fig. 1. Hierarchical structure of organizational units

In order to model the relationships between the different peer types the UML notation for aggregation, composition and association is used (cf. figure 1):

- **Aggregation** and **composition** describe a close cooperation between organizational units or peers, respectively, and can also be used to describe their hierarchical order. Typically, to establish an aggregation or composition relationship a peer registers at a superior peer. The superior peer has the special ability to control its registered peers. In figure 1 for example group peer G_1 has control over the single peers S_3 to S_5. The group peers G_2 and G_4 are connected by a composition relationship. This means that peer G_4 cannot exist without group peer G_2 whereas in an aggregation relationship the partners of the aggregation can exist without each other.
- The **association** describes a loosely coupled cooperation with widely autonomous partners. No clear hierarchical structure can be extracted from an association relationship. The groups G_1 and G_2 in figure 1 cooperate. The same is true for the developers within these groups so that access to resources

of the respective other group can be granted to them. The developer at single peer S_9 is associated to group peer G_3 which means that he cooperates with the group G_3. Since an association signifies a loose connection only, a developer can be associated to more than one group at a time.

3 Resource Integration

For cooperative software development developer groups typically use a number of tools like version control systems (e.g. CVS or subversion) and CSCW systems. In the following such tools, storages, or services are understood as *resources*. In the context of a tight cooperation organizational units share resources among each other. A project peer for example usually shares its resources with the involved groups, i.e. the group peers are granted access to these resources. Group peers typically share their resources with single peers, i.e. all developers in a project can use the resources which are connected to the appropriate project peer.

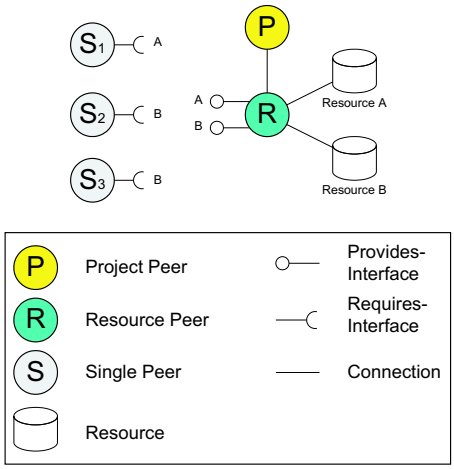


Fig. 2. Resource peer

Resources can directly be integrated into a peer-to-peer network as can be seen in figure 2. The resource peer R connects to the resources A and B in an application specific way. Externally another interface (e.g. in form of a web service) is offered to the rest of the peers. The peers connect to a resource through a special adapter which uses the corresponding external interface of the resource peer. The resource peer either manages access control lists of the peers which have access to the resources or queries another peer, like the project peer P in figure 2, whether to grant access to the requesting peer. All in all, the following steps are necessary to connect to a resource which is integrated into a peer-to-peer network:

1. The requesting peer (e.g. S_2) asks the resource peer for access to a resource.
2. The resource peer asks the responsible peer for access rights or consults its access control list.
3. Afterwards, access is granted or denied to the requesting peer.
4. If access is granted, the resource can be accessed.

By using the interfaces of a resource peer other peers can for example access documents which are stored on a CVS server or a local device. Another example for the shared use of resources is the registration of group members for a forum or a groupware system carried out by their group peer via a resource peer.

4 Multi-tier Look Up Service

The lookup service is a central requirement for peer-to-peer systems. It assigns and locates resources and artifacts among peers [3]. Distributed "flat" peer-to-peer lookup services are e.g. Chord [4], CAN [5], Pastry [6] and Tapestry [7]. The approach presented in this paper is the introduction of a hierarchical multi-tier lookup service where peers are organized in disjoint clusters as it is depicted in figure 3. Super peers route the messages along clusters to the destination cluster. Within the clusters the messages move through the hierarchical structure of the peers. The hierarchical peer structures in combination with their super peers form a hierarchical super peer network [8]. The super peers hold a common metadata index of available artifacts which are distributed over the different organizational units or peer types, respectively. They are able to answer simple queries. Detailed queries additionally pass through the hierarchical structure of the peers. The exchange of the located artifacts takes place directly from peer to peer.

Super peer networks have some advantages in comparison to pure peer-to-peer networks. They combine the efficiency of the centralized client-server model with the autonomy, load balancing, and robustness of distributed search. They also take advantage of the heterogeneity of capabilities across peers. Parameters like cluster size and super peer redundancy have to be considered by designing a super peer network. Redundancy decreases individual super peer load significantly whereas the cluster size affects the behavior of the network in an even more complex manner [9, 10].

The most important benefits of the approach discussed in this paper are scalability and administrative autonomy. A super peer can independently route messages within its cluster. Similarly, organization, project, and group peers can route the messages to subordinated peers using their own strategy. Queries to selected organizational units do not flood the entire network, but can be routed directly.

4.1 Metadata Index

The super peers are connected to each other and share a common metadata index of the available artifacts. Physically, any peer which has enough computing

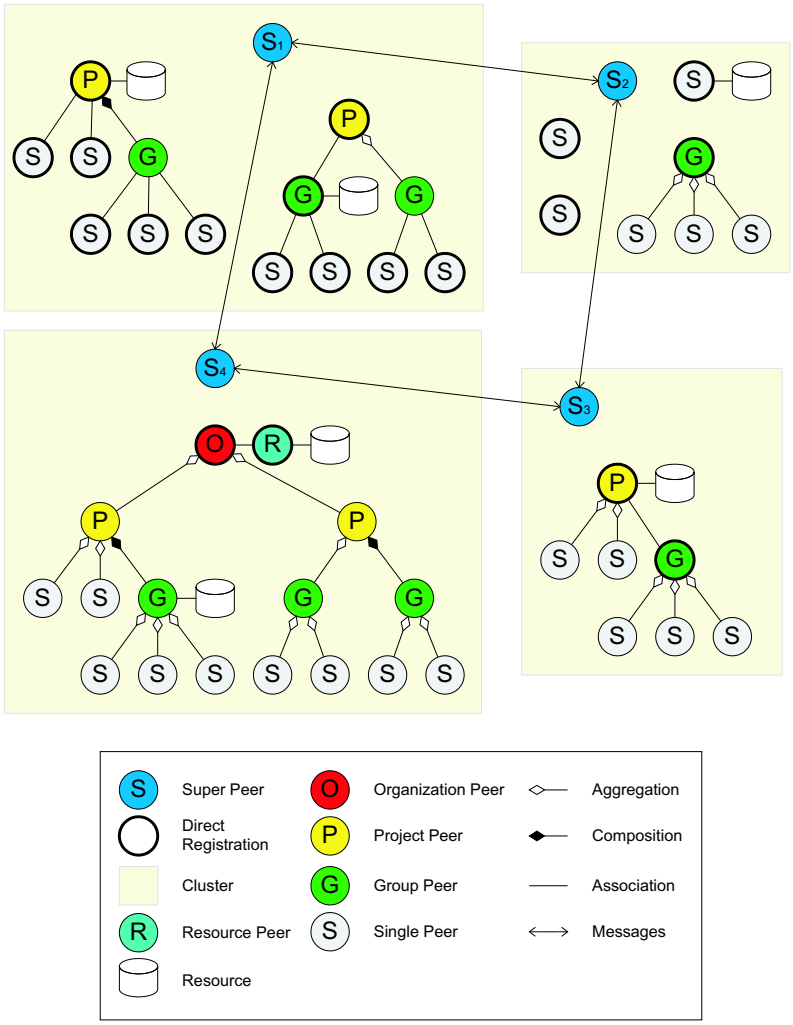


Fig. 3. Multi-tier lookup service in a hierarchical super peer network

power, storage capacity, and an adequate network connection can be chosen to be a super peer. In figure 3, a ring of super peers is shown for reasons of simplicity. There are other techniques like HyperCuP [11] that are able to increase the scalability and reduce the lookup time drastically. Peers which are registered at a super peer make up a cluster as depicted in figure 3.

There are different ways in which super peers can collect metadata. Typically peers which are linked to other peers by associations only register at the associated peers and send their metadata directly to the super peer whereas peers which are in an aggregation or composition relationship register at the superior peer and send their metadata there. Thus, a superior peer has extensive knowledge of the subordinated peers. The superior peers send the available metadata to higher peers in the hierarchical order until the super peer is reached. One advantage of this approach is that not every peer in a network needs an internet connection to make its metadata available. Furthermore, the superior peers have control over the metadata which is sent to a super peer. Hence, superior peers have the ability to decide whether metadata of subordinated peers is made available or unavailable to superior peers or the global index, respectively.

Within the metadata index additional information on the organizational units like group members, project goals and capabilities of developers is stored. Moreover, metadata can also be extracted from resources which are connected to a resource peer. In figure 3 a simplified view of resource connections is illustrated in which the resource peer is not explicitly visible. In this case the associated peers are assumed to have the capabilities of a resource peer.

5 A General Super Peer Network for Digital Libraries

The hierarchical super peer network for distributed software development described above can be generalized to support the flexibility and self-organization of widely distributed, loosely coupled, and autonomous digital library systems. The architecture allows for the search over collections of arbitrary artifacts as for example traditional documents, on-line books, digital images, and videos, which is a basic service requirement for digital libraries [12]. Beyond, the network enables library users to also store, administer, and classify their own artifacts. Thus, it supports scenarios like the construction of personal or group reference libraries and collaborative authoring.

Figure 4 depicts a hierarchical super peer network for digital libraries. The organizational units and their respective peer types are adapted to the situation within a general digital library. Persons are able to search for artifacts and offer artifacts on their own. They are therefore supplied with *person peers*. On the next organizational level, the artifacts are grouped within collections managed by *collection peers*. Collection peers offer functionality relating to collection organization as for example the provision of a common classification scheme. A digital library can combine a number of different collections and is associated with a *digital library peer*. A digital library peer supports the integration of different collections, for example by offering merging services for different classification

schemes [13]. Furthermore, it manages access to the digital library artifacts, for example by ensuring a certain mode of payment [14]. Person peers and collection peers can also exist independently from a superior peer and autonomously offer artifacts.

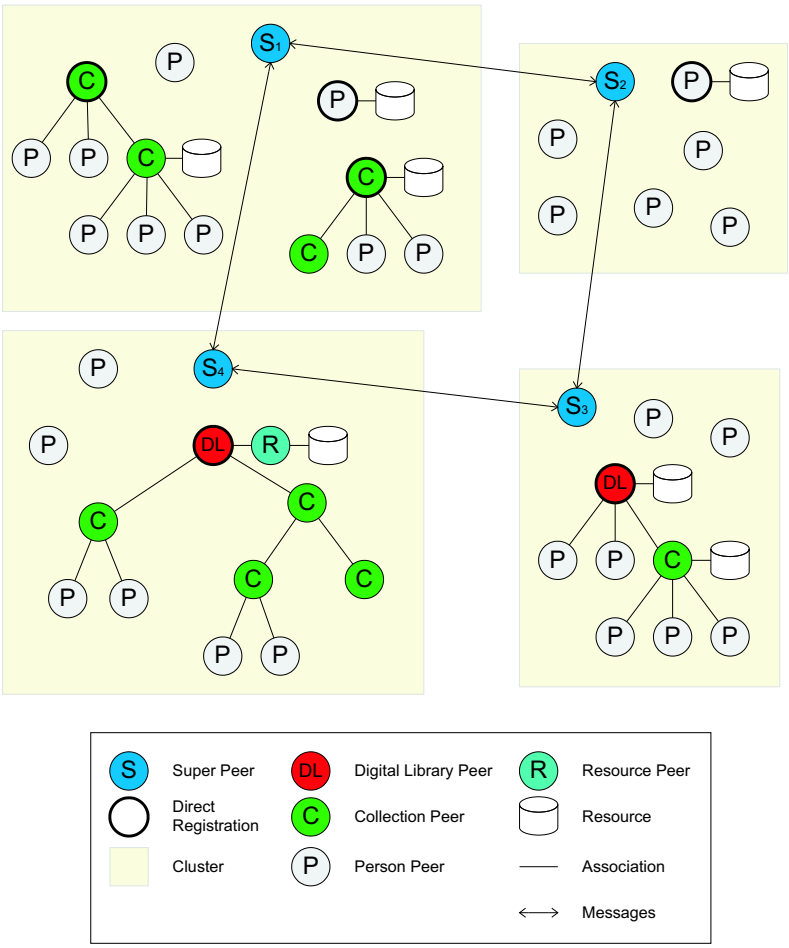


Fig. 4. A hierarchical super peer network for digital libraries

Person peers, collection peers, and digital library peers, as in the approach for distributed software development presented above, are clustered. The clusters again are connected via super peers in the described manner (cf. sect. 4) and searched via super peers and superior peers.

The described network represents a first step in order to capitalize on the advantages of peer-to-peer technology for digital libraries. For personal or project

reference libraries most of the upcoming traffic will remain within subareas of the network where co-workers cooperate intensely. For specialized collections which focus on special topics or special media types queries can be routed directly to selected collections or even library experts without flooding the entire network. Precision and query performance can hence be improved. Additionally, a self-organization of collections and libraries is possible. Scalability and administrative autonomy are also ensured.

6 Related Work

Enabling interoperability among heterogeneous, widely distributed, autonomous digital library services is also the purpose of some other projects as described for example in [12]. The goal of establishing a manageable system of personal and project reference libraries as pursued in [1] also calls for a flexibility which can be achieved by the use of a super peer network.

The design of a super peer network is described in [9]. The costs and benefits of a new hybrid approach called structured super peers is explored in [10]. It partially distributes lookup information among a dynamically adjusted set of high-capacity super peers to provide constant-time lookup. Super peer based routing strategies for RDF-based peer-to-peer networks are described in [15].

In hierarchical peer-to-peer systems, peers are organized into groups, and each group has its autonomous intra-group overlay network and lookup service. A general framework and scalable hierarchical overlay management is provided in [8].

7 Conclusions and Future Work

This paper has presented a super peer network approach for autonomous and self-organizing digital libraries and artifact collections and substantiated it by describing a network instance for special libraries dedicated to software development tasks.

One future challenge with regard to hierarchical super peer networks is to analyze the dynamic behavior of the network, particularly if peers fail. Enhancing the availability of artifacts and peer services by replication seems to be one promising approach to solve this problem.

Another issue is to gain further understanding on how the presented approach can be refined against the background of reference libraries. The use of project peers as they have already been introduced for distributed software development could be an option. Yet, the project peers have to be adequately fit into the overall hierarchical structure of the network.

References

1. Schmidt, J.W., Schröder, G., Niederée, C., Matthes, F.: Linguistic and Architectural Requirements for Personalized Digital Libraries. International Journal of Digital Libraries **1** (1997)

2. Schollmeier, R.: A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications. In: 1st International Conference on Peer-to-Peer Computing (P2P 2001), Linköping, Sweden, IEEE Computer Society (2001)
3. Balakrishnan, H., Kaashoek, M.F., Karger, D., Morris, R., Stoica, I.: Looking Up Data in P2P Systems. Communications of the ACM **46** (2003) 43–48
4. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord - A Scalable Peer-to-peer Lookup Service for Internet Applications. In: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, ACM Press (2001) 149–160
5. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Schenker, S.: A Scalable Content-Addressable Network. In: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, ACM Press (2001) 161–172
6. Rowstron, A., Druschel, P.: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In: IFIP/ACM International Conference on Distributed Systems Platforms (Middleware). (2001) 329–350
7. Zhao, B.Y., Huang, L., Rhea, S.C., Stribling, J., Joseph, A.D., Kubiatowicz, J.D.: Tapestry: A Resilient Global-Scale Overlay for Service Deployment. IEEE J-SAC **22** (2004) 41–53
8. Garces-Erice, L., Biersack, E.W., Ross, K.W., Felber, P.A., Urvoy-Keller, G.: Hierarchical Peer-to-peer Systems. In: Proceedings of ACM/IFIP International Conference on Parallel and Distributed Computing (Euro-Par), Klagenfurt, Austria (2003)
9. Yang, B., Garcia-Molina, H.: Designing a Super-Peer Network. In: IEEE International Conference on Data Engineering, 2003, San Jose, California (2003)
10. Mýzrak, A.T., Cheng, Y., Kumar, V., Savage, S.: Structured Superpeers: Leveraging Heterogeneity to Provide Constant-Time Lookup. In: The Third IEEE Workshop on Internet Applications, San Jose, California (2003)
11. Schlosser, M.T., Sintek, M., Decker, S., Nejdl, W.: HyperCuP - Hypercubes, Ontologies, and Efficient Search on Peer-to-Peer Networks. In Moro, G., Koubarakis, M., eds.: Agents and Peer-to-Peer Computing, First International Workshop, AP2PC 2002, Bologna, Italy, July, 2002, Revised and Invited Papers, Springer (2002) 112–124
12. Paepcke, A., Chang, C.K., Gravano, L., Baldonado, M.: The Stanford Digital Library Metadata Architecture. (1997)
13. Matthes, F., Niederée, C., Steffens, U.: C-Merge: A Tool for Policy-Based Merging of Resource Classifications. In: Research and Advanced Technology for Digital Libraries, Proceedings of the 5th European Conference, ECDL2001, Darmstadt, Germany. (2001)
14. Weber, R.: Chablis - Market Analysis of Digital Payment Systems. Institutsbericht, Technische Universitaet Muenchen, Institut fuer Informatik (1998)
15. Nejdl, W., Wolpers, M., Siberski, W., Schmitz, C., Schlosser, M., Brunkhorst, I., Löser, A.: Super-Peer-Based Routing Strategies for RDF-Based Peer-to-Peer Networks. Web Semantics: Science, Services and Agents on the World Wide Web **1 Issue 2** (2004) 177–186

An Information Service Architecture for Annotations

Maristella Agosti and Nicola Ferro

Department of Information Engineering – University of Padua
Via Gradenigo, 6/b – 35131 Padova (PD) – Italy
{maristella.agosti, nicola.ferro}@unipd.it

Abstract. This paper presents the architecture and the features of an annotation service for digital libraries. Firstly, it describes the relevant aspects of annotations and the different way of using them. Then it shows how the characteristics of the annotations drive the design choices of the architecture. Finally it discusses the design choices and the architectural alternatives for developing the annotation service.

1 Introduction

The main objective of this research is to create an Annotation Service (AS) for Digital Libraries (DL), which deals with many aspects of annotations, such as creation, management, access and retrieval of both manual and automatically created annotations. The service at hand is innovative because in fact it is able to actively exploit annotations in order to select and retrieve documents from a digital library in response to a user query.

Over past years a lot of research work regarding annotations has been done [18]: user studies for understanding annotation practices and discovering common annotation patterns [12–14, 22]; investigation and categorization of various facets of the annotation [3, 8, 13]; employment of ad-hoc devices or handheld devices which enable reading appliances with annotation capabilities [15–17, 21]; design and development of document models and applications which support annotations in digital libraries, in the Web, in collaboratory systems and working groups [3, 8, 10, 18, 20, 23, 24]. All this research work has led to different viewpoints about what annotations are:

- *annotations are metadata*: they are additional data about an existing content. For example, the World Wide Web Consortium (W3C) [10, 23, 24] considers annotations as metadata and interprets them as the first step in creating an infrastructure that will handle and associate metadata with content towards the Semantic Web;
- *annotations are contents*: they are additional content about an existing content; they increase existing content and allow the creation of new relationships among existing contents, by means of links that connect annotations together and with existing content. In this sense we can consider that existing

content and annotations constitute a hypertext, according to the definition of hypertext provided in [1]. For example, [13] considers annotations as a natural way of enhancing hypertexts by actively engaging users with existing content in a digital library [12, 15, 21];
- *annotations are dialog acts*: they are part of a discourse with an existing content. For example, [8] considers annotations as the document context, intended as the context of the collaborative discourse in which the document is placed.

In the following for annotation we mean *any piece of additional content associated with an existing content* and we consider that annotations and existing contents constitute a hypertext. Note that digital libraries do not usually provide a hypertext that links documents together; thus annotations can represent an effective means for creating a hypertext that links annotations and existing content together. This hypertext can be exploited not only for providing alternative navigation and browsing capabilities, but also for offering advanced search functionalities. Finally the hypertext existing between documents and annotations enables different annotation configurations, that are *threads of annotations*, i.e. an annotation made in response or comment to another annotation, and *sets of annotation*, i.e. a bundle of annotations on the same passage of text. A comprehensive presentation on annotations and their facets can be found in [3, 4].

Furthermore annotations introduce a new content layer devoted to elucidate the meaning of an underlying information resource and they can make hidden facets of the annotated information resource more explicit. Thus, we can exploit annotations for retrieval purposes, and add the evidence provided by annotations themselves in order to better satisfy the user's information needs. For example, suppose that we have the following query: "good survey grid computing". A relevant document for this query could be ranked low because it does not contain the terms "good" and "survey". On the other hand, if the following annotation "good survey, which clearly explains the topic" is linked to the document, we can exploit it in order to better rank the document. Note that the annotation itself does not clearly states what topic is explained but this information can be still obtained exploiting the hypertext and navigating the link that connects the annotation to the document. Thus we can combine two distinct sources of evidence, the one coming from the document and the one coming from the annotation. This way the annotation and the document can cooperate together in order to better satisfy the user's information need. Indeed the combining of these multiple sources of evidence can be exploited to improve the performances of an information management system, i.e. to retrieve more relevant documents and to better rank them with respect to the case of a simple query without annotations. This is what we mean for "to actively exploit annotations in order to select and retrieve documents from a DL in response to a user query".

Finally, today the notion of isolated applications or data is increasingly disappearing in favour of a distributed and networked environment with an information centric view. This allows us to provide integrated services and applications to users, without any distinction between local and remote information resources.

In this context we can envisage a scenario in which a DL can become not only a place where information resources can be stored and made available, but also a daily work tool, which can be integrated into the way the user works, so that the user's intellectual work and contents which are provided by the digital library can be merged together, constituting a single working context. Thus the digital library is no longer perceived as something external to the intellectual production process or as a mere consulting tool but as an intrinsic and active part of the intellectual production process. Annotations are effective means used in enabling this paradigm of interaction between users and digital libraries for all the reasons previously explained and, in particular, because annotations allow users to naturally create an hypertext that seamlessly merges personal contents with the contents provided by the digital library.

The presentation of the findings is structured as follows: Section 2 introduces the design choices and the features of the architecture of the annotation service, while Section 3 describes each component of the annotation service and its functionalities.

2 Annotation Service Architecture

In the context introduced in Section 1, architectural choices become a key factor for enabling the design and development of an advanced annotation service capable of both modelling the different facets of the annotation and effectively exploiting annotations for search and retrieval purposes.

We need to design an architecture with a twofold aim: firstly, it has to model the behaviour of the AS in a modular way, so that we can easily add new functionalities to the AS without the need of redesigning the architecture of the AS. Secondly, the architecture has to be flexible enough to be implemented according to different architectural paradigms, such as Web Services (WS) or Peer–to–Peer (P2P) architectures (figure 1). Indeed a flexible architecture allows the AS to have a great reach and a widespread usage, so that users can benefit from its functionalities without limitations due to the architecture of a particular DL, allowing a strict interaction between users and digital libraries.

Thus the annotation service has to comply with the following constraint: it must be able to work with diverse DL systems.

We introduce this constraint for three reasons:

- annotations are a key technology for actively engaging users with a DL. Users need to have an annotation service easily available for each DL they work with and they should not change their annotative practices only because they work with a different DL system. Thus the annotation service must be capable to be integrated into various DL systems in order to guarantee a uniform way of interaction to the users;
- annotations create a hypertext that allow users to merge their personal content with the content provided by diverse DL systems, according to the scenario envisage above. This hypertext can span and cross the boundaries

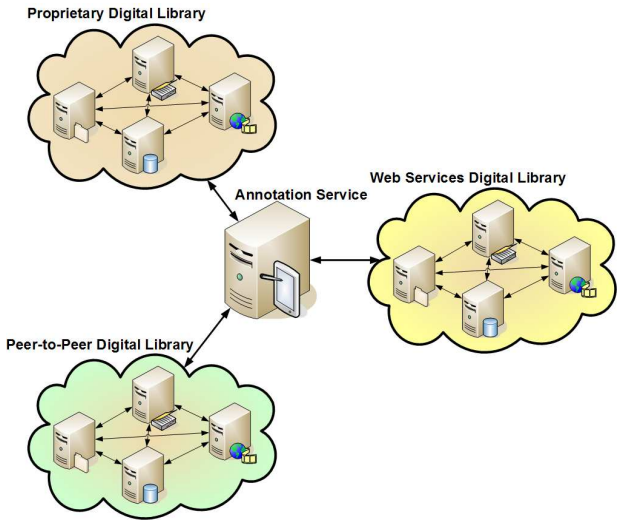


Fig. 1. Overview of the annotation service architecture.

of a single DL and thus the annotations service must be able to interact with diverse DL systems;
- annotations and information resources managed by the DL provide multiple sources of evidence for answering to a user's query. Since these sources of evidence can come from different DL systems and we need to combine them in order to answer the query, the annotation service must be able to work with diverse DL systems.

To comply with the constraint just defined, we have decided to adopt an architecture using a gateway which mediates between the DL and the core functionalities of the AS. Furthermore we have mapped this choice into a three layer architecture, which allows us to have a better modularity. Figure 2 demonstrates this architecture, where the AS is depicted on the right, and in the cloud the external DL system is represented.

The architecture is organized along two dimensions:

- *vertical decomposition* (from bottom to top): consists of three layers – the data, application and interface logic layers – and it is concerned with the organization of the core functionalities of the AS.
  This decomposition allows us to achieve a better modularity within the AS and to properly describe the behaviour of the AS by means of isolating specific functionalities at the proper layer. Moreover it makes it possibile to clearly define the workflow within the AS by means of communication paths that connect the different components of the AS itself. This way we can achieve the first aim of our architecture, that is to model the behaviour of the AS in a modular way.

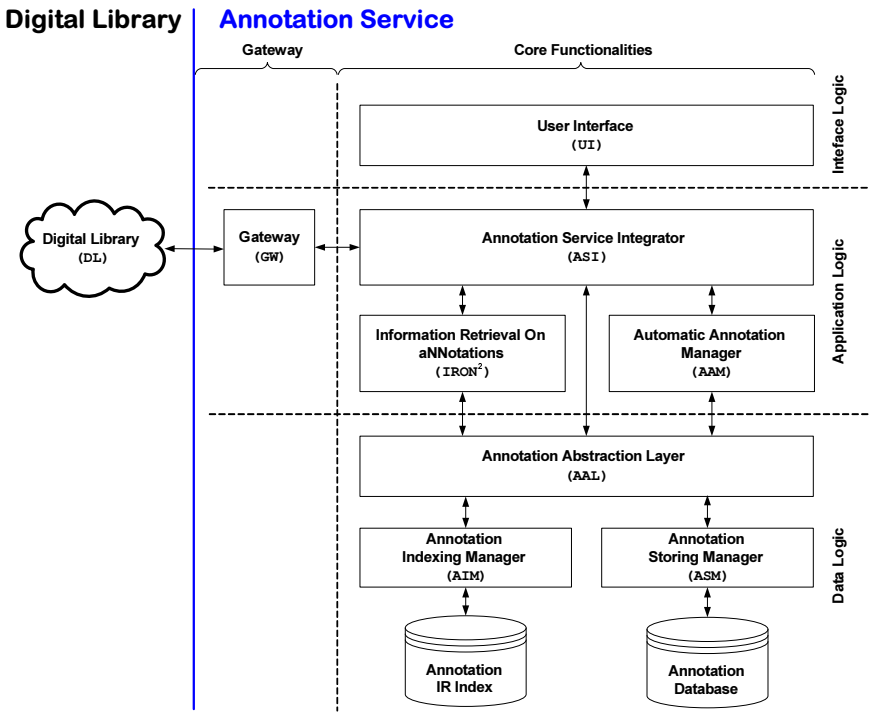


Fig. 2. Annotation Service Architecture.

– *horizontal decomposition* (from left to right): consists of the DL system, the gateway and the core services of the AS. It separates the core functionalities of the AS from the problem of integrating the AS into a specific DL system. The horizontal decomposition allows us to achieve the second aim of our architecture, since we can integrate the AS with a different DL system simply by changing the gateway. Furthermore the architecture is presented in the context of a DL, but it is suitable for integration also with other systems that manage documents and offer information retrieval capabilities, such as a search engine or a collaboratory system [4].

In order to achieve a great flexibility, we design the architecture of the AS at a high level of abstraction, that is we define the functionalities of each component of the AS in terms of abstract Application Program Interface (API). This way we can model the behaviour and the workflow within the AS without worrying about the actual implementation of each component. Different alternative implementations of each component could be provided, still keeping a coherent view of the whole architecture of the AS.

On the whole, we have two levels of abstraction: the first level is the separation of the core features of the AS from the integration with the DL and the organization of the AS into different layers in order to clearly determine the

different functionalities of the AS; the second level is built on top of the first one and makes it possible to describe at a higher level of abstraction the behaviour of each component of the AS, separating its behaviour from the actual implementation of the AS. This way we can have multiple implementations of the AS all referring to the same architecture and API.

We achieve the abstraction levels described above by means of a set of interfaces, which define the behaviour of each component of the AS in abstract terms. Then, a set of abstract classes partially implement the interfaces in order to define the behaviour of each component. This way this behaviour becomes common to all of the implementations of that component. Finally, the actual implementation is left to the concrete classes, inherited from the abstract ones, that fit the AS into a given architecture, such as a WS or P2P architecture. Furthermore, we apply the *abstract factory* design pattern [9], which uses a factory class for providing concrete implementations of a component, compliant with its interface, in order to guarantee a consistent way of managing the different implementations of each component.

The AS is developed using the Java[1] programming language, which provides us great portability across different hardware and software platforms.

3 Components of the Annotation Service

In the following sections we describe each component of the AS, according to figure 2 from bottom to top.

3.1 Data Logic Layer

Annotation Storing Manager (ASM) manages the actual storage of the annotations and provides a persistence layer for storing the `Annotation` objects, that are used in the upper layers for representing annotations.

The ASM provides a set of basic operations for storing, retrieving, deleting and searching annotations in a SQL–like fashion. Furthermore it takes care of mapping `Annotation` objects into their equivalent representation for the actual storage, according to the Data Access Object (DAO)[2] and the Transfer Object (TO)[3] design patterns. This way all of the other components of the AS deal only with `Annotation` objects, which represent the TO of our system, without worrying about the details related to the persistence of such objects.

Annotations are modeled according to the Entity–Relationship (ER) schema of figure 3, which is described in detail in [4]. Briefly, the ER schema of figure 3 represents the fact that an Annotation must Annotate one and only one digital object, identified by its handle DOHandle, while it can Relate To one

[1] `http://java.sun.com`

[2] `http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html`

[3] `http://java.sun.com/blueprints/corej2eepatterns/Patterns/TransferObject.html`

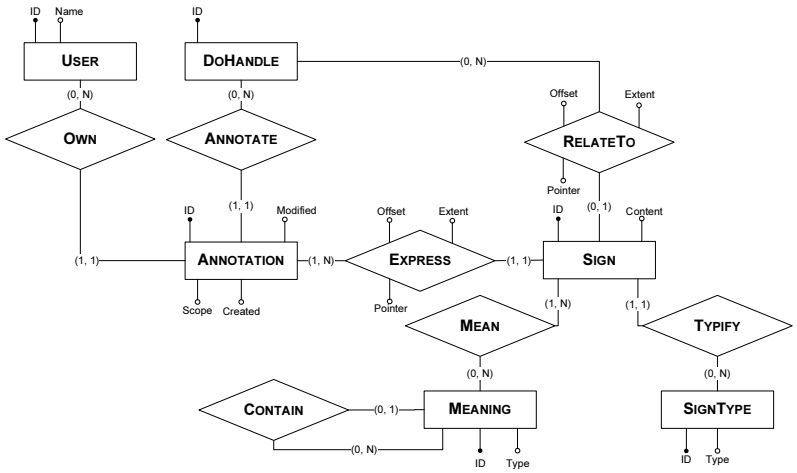


Fig. 3. Entity–Relationship schema for modelling annotations.

or more digital objects. Moreover one or more SIGNs, e.g. a piece of text or a graphic mark, contribute to EXPRESS an ANNOTATION and each SIGN can have one or more MEANINGs, that define its semantics.

Note that the ER schema of figure 3 can be easily mapped to different designing models, such as a relational schema, a Resource Description Framework (RDF) schema or a eXtensible Markup Language (XML) schema. This way it gives us great flexibility with respect to different architectural choices and allows us to provide different implementations of the ASM in a completely transparent manner for the other components of the AS.

Annotation Indexing Manager (AIM) provides a set of basic operations for indexing and searching annotations for information retrieval purposes.

The AIM is a full-text information retrieval system and deals with the textual content of an annotation. It is based on the experience acquired in developing IRON (Information Retrieval ON), the prototype information retrieval system that we have developed to be used in the Cross Language Evaluation Forum (CLEF) evaluation campaigns since 2002; the main functionalities of IRON are described in [2, 7].

Annotation Abstraction Layer (AAL) abstracts the upper layers from the details of the actual storage and indexing of annotations, providing uniform access to the functionalities of the ASI and the ASM.

The AAL provides the typical Create–Read–Update–Delete (CRUD) data management operations, taking care of coordinating the work of the ASM and the ASI together. For example when we create a new annotation, we need to put it in both the ASM and in the ASI or, when we delete an annotation, we need to remove it from both the ASM and the ASI.

Furthermore the AAL provides search capabilities forwarding the queries to the ASM or to the AIM. At the moment there is only the AIM for providing full text search capabilities but, in the future, other specialised information retrieval engines can be added to the system, for example for indexing and searching the graphical content of an annotation and other types of digital media. In any case the addition of other information retrieval engines becomes transparent for the upper layers, since the AAL provides uniform access to them.

Note that both the ASM and the AIM are focused on each single annotation in order to properly store and index it, but they do not have a comprehensive view of the relationships that exist between documents and annotations, that is they do not take into consideration the hypertext mentioned in Section 1. On the contrary, the AAL has a global view of the annotations and their relationships and exploits it for managing purposes. For example, if we delete an annotation that is part of a thread of annotations, what policy do we need to apply? Do we delete all the annotations that refer to the deleted one or do we try to reposition those annotations? The ASM and AIM alone would not be able to answer this question but, on the other hand, the AAL can drive the ASM and the AIM to perform the correct operations.

Thus, on the whole, the AAL, the ASM and the AIM constitute an information management system specialised in managing annotations, as a database management system is specialised in managing structured data.

3.2 Application Logic Layer

Information Retrieval on aNNotations (IRON^2) provides advanced search capabilities based on annotations, such as those previously introduced.

As an important consequence of our architecture, we know everything about annotations but we have no knowledge about documents managed by the DL. This is due to the fact that we directly manage annotations while documents and information pertaining to them are provided by the DL.

This architectural choice influences the way in which our search strategy is carried out and retrieving documents by using annotations involves a complex strategy. Firstly, the AS receives a query from the end-user, the query is used to select all the relevant annotations, an annotation hypertext can now be built and used to identify the related documents. Now we aim to combine the source of evidence coming from annotations with the one coming for the documents managed by the DL, as previously explained. Since the source of evidence concerning the documents is completely managed by the DL, the AS has to query the DL, which gives back a list of relevant documents. Only after that the AS has acquired this information from the DL, it can combine it with the source of evidence coming from annotations in order to create a fused list of result documents that are presented to the users to better satisfy his information needs.

The Unified Modeling Language (UML) sequence diagram [19] of figure 4 shows how this search strategy involves many of the components of the AS.

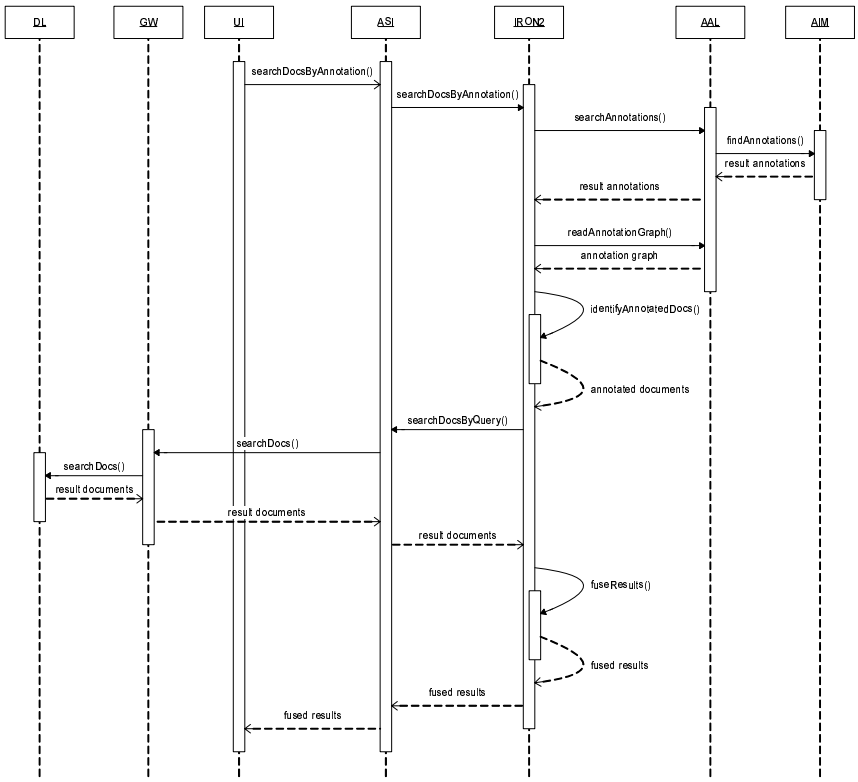


Fig. 4. Sequence diagram for searching documents exploiting annotations.

Automatic Annotation Manager (AAM) creates automatic annotations for a given document.

Automatic annotations can be created using topic detection techniques to associate each annotation with its related topic, which forms the context of the annotation. In this way a document can be re-organized and segmented into topics, whose granularity can vary, and annotations can present a brief description of those topics. Then by applying automatic hypertext construction techniques, similar to those presented in [5], document topics and annotations can be linked together, proposing an alternative way of navigating the content of a digital library.

Annotation Service Integrator (ASI) integrates the underlying components and provides uniform access to them. It represents the entry point to the core functionalities of the AS for both the gateway and the user interface, dispatching their requests to underlying layers and collecting the responses from the underlying layers.

The ASI can be further exploited for creating a network of P2P annotation services that cooperate together. In this scenario, when the ASI receives a request from the gateway or from the user interface – for example a search request – it forwards the request also to the other ASI peers and then collects their answers in order to provide access to the whole network of P2P annotation services.

Thus, our architecture allows us to implement the annotation service not only as a stand–alone service, that can be integrated into different DL systems, but also as a network of P2P annotation services that cooperate in order to provide advanced annotation functionalities to different DL systems.

Gateway (GW) provides functionalities of mediator between the core functionalities of the AS and the DL system. Simply by changing the gateway, we can share the same AS with different DL systems. We can envisage three kinds of gateway: firstly the AS could be connected to a DL which uses a proprietary protocol and in this case the gateway can implement it. This is the case, for example, of the OpenDLib digital library [6], with which the AS is going to cooperate [3]. Secondly we could employ Web Services to carry out the gateway, so that the AS is accessible in a more standardized way. Finally the gateway can be used to adapt the AS to a P2P digital library.

Note that the decoupling of the core functionalities of the AS from its integration within a specific DL is independent from the possibility of implementing the AS as a stand–alone service or as a network of P2P services. Indeed we can adapt both of these implementations to any kind of DL system by means of a proper gateway: in both cases the gateway represents the unique access point for the DL to the functionalities of the stand–alone AS or of the P2P network of AS.

3.3 Interface Logic Layer

User Interface (UI) provides an interface to end–users for creating, modifying, deleting and searching annotations. As show in figure 2 the UI is connected to the ASI, so that it represents the user interface of AS itself and it is independent of any particular DL system.

On the other hand, we can connect or integrate the UI directly in the gateway, so that it represents a user interface tailored to the specific DL for which the gateway is developed. In this case the gateway forwards the request of the UI to the ASI, for which the gateway acts also as user interface. For example, this choice has been adopted in integrating the AS into the OpenDLib digital library in order to obtain a user interface more coherent with those of the other OpenDLib services.

The design choice of connecting the UI to the gateway or to the ASI gives interesting possibilities for programmatically accessing the functionalities of the AS. As an example, if the gateway is implemented using Web Services, another service can programmatically access the AS in order to create more complex applications that exploit also annotations.

Acknowledgements

This work was partially funded by ECD (Enhanced Contents Delivery), a joined program between the Italian National Research Council (CNR) and the Ministry of Education (MIUR), under the law 449/97-99.

References

1. M. Agosti. An Overview of Hypertext. In M. Agosti and A. Smeaton, editors, *Information Retrieval and Hypertext*, pages 27–47. Kluwer Academic Publishers, Norwell (MA), USA, 1996.
2. M. Agosti, M. Bacchin, N. Ferro, and M. Melucci. Improving the Automatic Retrieval of Text Documents. In C. Peters, M. Braschler, J. Gonzalo, and M. Kluck, editors, *Evaluation of Cross–Language Information Retrieval Systems, Third Workshop of the Cross–Language Evaluation Forum, CLEF 2002, Revised Papers*, pages 279–290. Lecture Notes in Computer Science (LNCS) 2785, Springer, Heidelberg, Germany, 2003.
3. M. Agosti and N. Ferro. Annotations: Enriching a Digital Library. In Koch and Sølvberg [11], pages 88–100.
4. M. Agosti, N. Ferro, I. Frommholz, and U. Thiel. Annotations in Digital Libraries and Collaboratories – Facets, Models and Usage. In R. Heery and L. Lyon, editors, *Proc. 8th European Conference on Research and Advanced Technology for Digital Libraries (ECDL 2004)*. Lecture Notes in Computer Science (LNCS), Springer, Heidelberg, Germany (in print), 2004.
5. M. Agosti and M. Melucci. Information Retrieval Techniques for the Automatic Construction of Hypertext. In A. Kent and C.M. Hall, editors, *Encyclopedia of Library and Information Science*, volume 66, pages 139–172. Marcel Dekker, New York, USA, 2000.
6. D. Castelli and P. Pagano. OpenDLib: a Digital Library Service System. In M. Agosti and C. Thanos, editors, *Proc. 6th European Conference on Research and Advanced Technology for Digital Libraries (ECDL 2002)*, pages 292–308. Lecture Notes in Computer Science (LNCS) 2458, Springer, Heidelberg, Germany, 2002.
7. G. M. Di Nunzio, N. Ferro, M. Melucci, and N. Orio. Experiments to Evaluate Probabilistic Models for Automatic Stemmer Generation and Query Word Translation. In C. Peters, M. Braschler, J. Gonzalo, and M. Kluck, editors, *Evaluation of Cross–Language Information Retrieval Systems, Fourth Workshop of the Cross–Language Evaluation Forum, CLEF 2003, Revised Papers*. Lecture Notes in Computer Science (LNCS), Springer, Heidelberg, Germany (in print), 2004.
8. I. Frommholz, H. Brocks, U. Thiel, E. Neuhold, L. Iannone, G. Semeraro, M. Berardi, and M. Ceci. Document-Centered Collaboration for Scholars in the Humanities – The COLLATE System. In Koch and Sølvberg [11], pages 434–445.
9. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object–Oriented Software*. Addison-Wesley, Reading (MA), USA, 1995.
10. J. Kahan and M.-R. Koivunen. Annotea: an open RDF infrastructure for shared Web annotations. In V. Y. Shen, N. Saito, M. R. Lyu, and M. E. Zurko, editors, *Proc. 10th International Conference on World Wide Web (WWW 2001)*, pages 623–632. ACM Press, New York, USA, 2001.
11. T. Koch and I. T. Sølvberg, editors. *Proc. 7th European Conference on Research and Advanced Technology for Digital Libraries (ECDL 2003)*. Lecture Notes in Computer Science (LNCS) 2769, Springer, Heidelberg, Germany, 2003.

12. C. C. Marshall. Annotation: from Paper Books to the Digital Library. In R. B. Allen and E. Rasmussen, editors, *Proc. 2nd ACM International Conference on Digital Libraries (DL 1997)*, pages 233–240. ACM Press, New York, USA, 1997.
13. C. C. Marshall. Toward an Ecology of Hypertext Annotation. In R. Akscyn, editor, *Proc. 9th ACM Conference on Hypertext and Hypermedia (HT 1998): links, objects, time and space-structure in hypermedia systems*, pages 40–49. ACM Press, New York, USA, 1998.
14. C. C. Marshall and A. J. B. Brush. From Personal to Shared Annotations. In L. Terveen and D. Wixon, editors, *Proc. Conference on Human Factors and Computing Systems (CHI 2002) – Extended Abstracts on Human Factors in Computer Systems*, pages 812–813. ACM Press, New York, USA, 2002.
15. C. C. Marshall, M. N. Price, G. Golovchinsky, and B.N. Schilit. Introducing a Digital Library Reading Appliance into a Reading Group. In N. Rowe and E. A. Fox, editors, *Proc. 4th ACM International Conference on Digital Libraries (DL 1999)*, pages 77–84. ACM Press, New York, USA, 1999.
16. C. C. Marshall, M. N. Price, G. Golovchinsky, and B.N. Schilit. Designing e-Books for Legal Research. In E. A. Fox and C. L. Borgman, editors, *Proc. 1st ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL 2001)*, pages 41–48. ACM Press, New York, USA, 2001.
17. C. C. Marshall and C. Ruotolo. Reading-in-the-Small: A Study of Reading on Small Form Factor Devices. In W. Hersh and G. Marchionini, editors, *Proc. 2nd ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL 2002)*, pages 56–64. ACM Press, New York, USA, 2002.
18. K. Nagao. *Digital Content Annotation and Transcoding.* Artech House, Norwood (MA), USA, 2003.
19. Object Management Group (OMG). OMG Unified Modeling Language Specification – March 2003, Version 1.5, formal/03-03-01. `http://www.omg.org/technology/documents/formal/uml.htm`, last visited 2004, May 25.
20. T. A. Phelps and R. Wilensky. Multivalent Annotations. In C. Peters and C. Thanos, editors, *Proc. 1st European Conference on Research and Advanced Technology for Digital Libraries (ECDL 1997)*, pages 287–303. Lecture Notes in Computer Science (LNCS) 1324, Springer, Heidelberg, Germany, 1997.
21. B. N. Schilit, M. N. Price, and G. Golovchinsky. Digital Library Information Appliances. In I. Witten, R. Akscyn, and F. M. Shipman, editors, *Proc. 3rd ACM International Conference on Digital Libraries (DL 1998)*, pages 217–226. ACM Press, New York, USA, 1998.
22. F. Shipman, M. N. Price, C. C. Marshall, and G. Golovchinsky. Identifying Useful Passages in Documents based on Annotation Patterns. In Koch and Sølvberg [11], pages 101–112.
23. World Wide Web Consortium (W3C). Annotea Project. `http://www.w3.org/2001/Annotea/`, last visited 2004, May 25.
24. World Wide Web Consortium (W3C). EMMA: Extensible MultiModal Annotation markup language – W3C Working Draft 18 December 2003. `http://www.w3.org/TR/2003/WD-emma-20031218/`, last visited 2004, May 25.

Peer-to-Peer Overlays and Data Integration in a Life Science Grid

Curt Cramer, Andrea Schafferhans, and Thomas Fuhrmann

Institut für Telematik, Universität Karlsruhe (TH), 76128 Karlsruhe, Germany
{cramer|fuhrmann}@tm.uka.de
Lion Bioscience AG, 69123 Heidelberg, Germany
andrea.schafferhans@lionbioscience.com

Abstract. Databases and Grid computing are a good match. With the service orientation of Grid computing, the complexity of maintaining and integrating databases can be kept away from the actual users. Data access and integration is performed via services, which also allow to employ an access control.
While it is our perception that many proposed Grid applications rely on a centralized and static infrastructure, Peer-to-Peer (P2P) technologies might help to dynamically scale and enhance Grid applications. The focus does not lie on publicly available P2P networks here, but on the self-organizing capabilities of P2P networks in general. A P2P overlay could, e.g., be used to improve the distribution of queries in a data Grid.
For studying the combination of these three technologies, Grid computing, databases, and P2P, in this paper, we use an existing application from the life sciences, drug target validation, as an example. In its current form, this system has several drawbacks. We believe that they can be alleviated by using a combination of the service-based architecture of Grid computing and P2P technologies for implementing the services.
The work presented in this paper is in progress. We mainly focus on the description of the current system state, its problems and the proposed new architecture. For a better understanding, we also outline the main topics related to the work presented here.

1 Introduction

There are many cases in which the integration of data from different sources gains additional value [13]. This is why most of the proposed applications of Grid Computing have data integration at their core.

In the Grid community, it is a common assumption that certain sets of Grid participants form *virtual organizations* (VOs) that exist only for a limited time and aim at jointly achieving a certain, well-defined goal [5]. The members of a VO, however, may be limited in their rights to access data from other members of their VO. Therefore, data access should be exposed to VO members only via services. The access control is then performed on the basis of these services.

A VO could, e.g., be formed by different life science research laboratories, both in universities and in companies. As continuous example in this paper, we

use a typical application from the life sciences, namely validation of drug targets. Here, the involvement of a protein in a disease and the potential of treating the disease by activating or inhibiting the protein is evaluated with a variety of biochemical and in-silico methods.

The potential targets and the data gathered in validating them are the intellectual property of the respective pharmaceutical companies and their contracted biotech partners. These companies typically maintain huge database libraries containing information about expression of the proteins in different tissues, protein-protein interactions, interactions between proteins and small molecules (potential drugs), etc. Some of the relevant information, e.g. protein and gene sequences, is publicly available from different research institutions, but comes in a variety of formats, often flat files due to historical reasons.

Today, pharma companies replicate the contents of the publicly available databases to local repositories, where they are pre-processed and integrated with the in-house data to allow complex queries while ensuring that proprietary information does not leave the company. This means, however, that data not replicated in the in-house repository cannot be accessed. Furthermore, many pharma companies outsource biochemical analyses to specialised biotech companies (e.g. profiling the interactions of a target protein or solving 3D structures) and need to integrate their results with the in-house data. Finally, today's globally operating pharma companies are faced with the problem of integrating data from their various sites, with often separately evolved database systems.

Here, Grid Computing with its service orientation is a promising approach to reduce the database maintenance overhead of individual institutions. Ideally, the required external databases could be remotely queried in their up-to-date state without the need to repeatedly copy and pre-process them locally. E.g., database services could be defined with the OGSA-DAI toolkit [10] and then be used to integrate the different data sources.

Collaboration of separate companies within a VO is very unlikely to result in open sharing of data between different parties. Especially biotech companies that perform services for different pharma companies need to give their individual clients access only to specific subsets of their results. Furthermore, since information gathered about a target protein is confidential, it should not be accessible to competitors. Pharma companies do not want to let their competitors know about the specific research performed by deducing patterns from the queries made by them. Hence some kind of "query obfuscation" needs to be employed to reduce this risk.

In order to protect the intellectual property of a company, all queries have to be executed on local database servers. To relieve the database servers from the high load posed on them, distributed query evaluation is needed. There are two extreme types of distributed evaluation. First, with *data shipping*, all data relevant to a query is transferred to one node which then has the burden to do all the processing associated with the query and afterwards return the result. *Query shipping*, on the other hand, transfers (parts of) the query to the data sources, where intermediary results are generated. These intermediary results

are exchanged between the data sources in order to construct the final result. Mixed forms of these two approaches are conceivable.

As mentioned above, shipping data to third parties might be problematic, since companies do not want their competitors to access their proprietary and sensitive data. Therefore, in this paper, we present our ongoing work to evaluate architectures for query shipping which allow to reduce the processing load put on the nodes. The approach we are currently following is to use a P2P overlay for scheduling queries on node subsets of VOs. The constitution of these subsets is controlled by particular companies, which is why the subsets can be considered trustworthy. The size of the subsets can be adapted dynamically to scale the system with increasing load, since the nodes in the subsets organize themselves in a P2P manner.

In order to provide interoperability with existing applications, our P2P query execution mechanisms on these subsets will be exported as a standard Grid service, e.g. using the interfaces defined by OGSA-DAI. This hides the functionality of query scheduling behind OGSA-DAI's standardized interfaces. By doing this, we can also employ the service description methods of OGSA-DAI to semantically integrate different data sources using appropriate ontologies.

The paper is structured as follows: In section 2, we describe the current architecture of the application example explained above. Then, we point out the major issues with it. After outlining the work related to ours in section 3, section 4 describes and discusses the architecture of an improved system which is to remove the previously mentioned issues. We then conclude in section 5 by highlighting the future directions of our work.

2 Problem Definition

In the last section, we gave a general overview of the drug target validation application. The currently employed architecture of this system is depicted in figure 1. Each pharma company replicates (some of) the content stored in public databases into its own database. Additionally, data generated by contracted third parties is merged into the local database. This data is only to be used by the respective pharma company. Note that although depicted as if the data were sent over the Internet, it is quite common to send these data on CDs to the pharma companies. Both the data retrieved from public databases and from third party contractors are manually integrated with the proprietary data of the pharma companies. Then, in-silico methods can be applied to the data.

This simple and pragmatic solution has several drawbacks:

1. Since the public databases have a size of several gigabytes, the data transfer takes a long time. Additionally, bandwidth and storage are aggessively consumed by the repeated replication of the full databases, even if only parts of them are actually accessed. Likewise, even databases that are only rarely accessed need to be replicated to the local system in order to be integrated with the other databases.

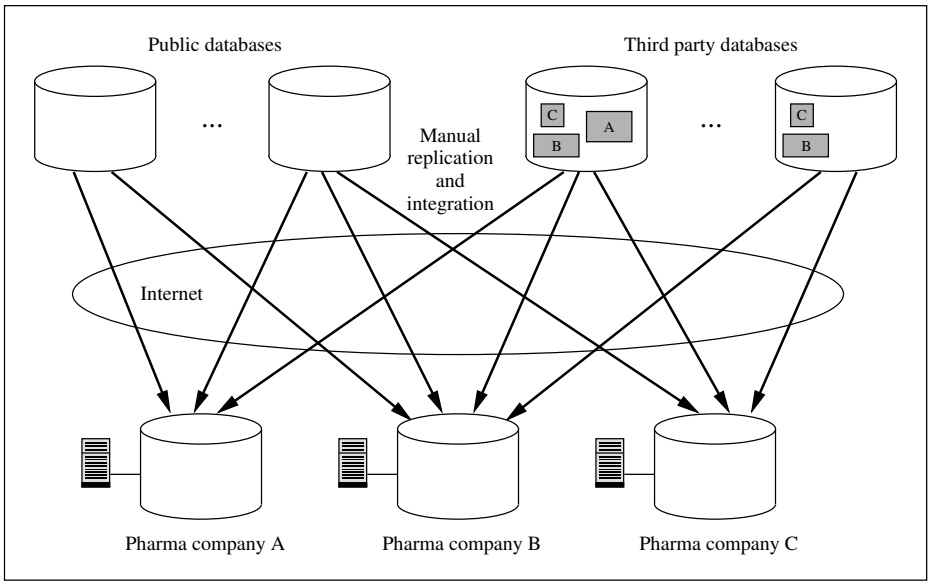


Fig. 1. Current System Architecture

2. Content stored in the public databases changes frequently. Hence, even more bandwidth and processing power is consumed by constant updating. Furthermore, it sometimes occurs that the data is outdated once the in-silico calculations have finished. The full cycle then has to be repeated.

We aim at eliminating these drawbacks while retaining the strong confidentiality constraints posed on the system: No proprietary data is allowed to leave a company and no other party than the pharma company itself must know about the analyses performed on the data.

Note that these security requirements are implicitly fulfilled in the system depicted in figure 1: All computations are performed at the respective pharma companies. If a third party were to do these computations, it might deduce some knowledge about the projects run by the pharma companies. We do not want the third party contractors or the public database providers to be able to gain insight into the intellectual property of a pharma company. This includes relations between facts that could be derived from query patterns.

3 Related Work

The work presented in this paper is related to several research areas. In this section, we point out their main characteristics and especially those relevant to our work.

3.1 Grid Computing

The term *Grid Computing* is associated with the sharing of resources in virtual organizations (VOs) formed by multiple institutions [5]. A VO is formed for following a common goal which is to be jointly achieved. As in real life, the participation of institutions is constrained by rules. Example applications of Grid computing include manufacturing, where each supplier in the chain is member of the VO, high energy physics with its vast storage requirements and many more. Grid applications typically have vast processing or storage requirements.

Grid computing employs a service-oriented approach and is heavily focused on standardization [4]. Services are defined and accessed by using standardized languages. Since all services use the same set of standards and languages, convenient service discovery through registries is enabled. Participants of a VO might be limited in their rights to access certain resources and therefore resource access has to be restricted using security services [4].

As discussed by Watson [13], the combination of different data sources provides added value. Many applications rely on databases, however existent database management systems (DBMSs) are not especially designed for Grid environments. Since huge investments have been made for the development of DBMSs and for maintaining the data, one cannot simply develop new DBMSs for Grids. Existent DBMSs have to be integrated with the Grid [13].

Kunszt et al. [9] discuss how databases can be fit into the standards and the Grid service model. An example implementation for wrapping commodity databases into Grid services has been produced in the OGSA-Database Access and Integration project (OGSA-DAI) [13, 10]. This toolkit is easy to use and adapt to new requirements.

3.2 Peer-to-Peer Overlays

Peer-to-Peer (P2P) networks are powerful way to enable resource sharing between hosts, called peers here. The peers connect to each other using certain rules for creating a virtual network topology which is overlaid to the underlying physical network. Links in this topology are realized as transport channels while the P2P protocols are situated at the application layer in the layer model of the Internet.

Early P2P overlays were so-called *unstructured* P2P networks. There, the formation of the overlay topology is very flexible and does not prescribe the choice of overlay links or the number of links. The flexibility of unstructured P2P networks allows nodes to choose "good" links, e.g. with low delay. However, since the topology is random, data items cannot be deterministically located. This is the reason why unstructured P2P networks like Gnutella [8] employ flooding or random walks in order to locate items.

To overcome the scalability and efficiency problems of flooding-based P2P protocols, structured overlay networks, e.g. Distributed Hash Tables (DHTs), have been proposed. There, the topology formation rules are quite rigid to assure routing efficiency. Each node is assigned a pseudo-random identifier, based on

which packet routing is done in the overlay. Nodes have to choose their links according to a pre-defined scheme in order to provide routing guarantees. I.e., a node commonly maintains $O(\log n)$ links and thus the average lookup path length in the overlay also is $O(\log n)$, where n is the number of nodes participating in the overlay. A widespread example of structured overlays is Chord [12].

The virtualized topology of structured overlays commonly does not match with the topology of the underlying physical network. Therefore, latencies for sending messages in the overlay are a multiple of the latency observed when sending the message between the source and sink in the underlay. This multiple is called the *relative delay penalty (RDP)*. A lot of work has been done on reducing the RDP in structured overlays, however success is limited by the respective type of the overlay [6].

A lot of applications have been proposed to be run on P2P overlay networks. In unstructured overlays, file sharing was a typical application. Research on structured overlays also started with this application type, however there has been a shift to other applications like multicast recently. In this light, we see P2P overlays as a powerful technology to perform application-specific routing independent of some physical network constraints.

As was already mentioned by Foster [5, 3], current P2P systems have not yet reached the state where protocol integration and interoperability are desired. However, both Grid computing and P2P systems are concerned with the sharing of resources and have similar objectives [3]. Therefore, the combination of insights from both approaches seems to be promising.

3.3 Distributed Database Systems

In the described application, there is a multitude of databases which are to be *integrated* in order to run combined queries over them. Since these databases are individually controlled and have different schemas — they in fact can also use different data models — the system can be seen as a *distributed multidatabase system* [11].

There are several points to be handled in distributed multidatabase system (Distributed Multi-DBMS). If the Multi-DBMS should have its own global schema, the schemas of the individual databases have to be integrated. This is a very complex task. We do not consider schema integration here since we assume that the data coming from different databases are disjoint and queries mainly involve joining them. There, the attributes to join on are already specified in the queries.

Another point which is not of interest here is transaction management. We assume that the public and the third party databases cannot be modified by the companies using them. Therefore, no serialization problems can occur.

Apart from schema integration and transaction management, distributed query execution is another important aspect of Distributed Multi-DBMSs. As explained by Özsu et al. [11], a Distributed Multi-DBMS can be seen as a layer on top of the individual DBMSs. Each individual database runs an instance of this layer. This layer can be seen as a wrapper which exports the database's

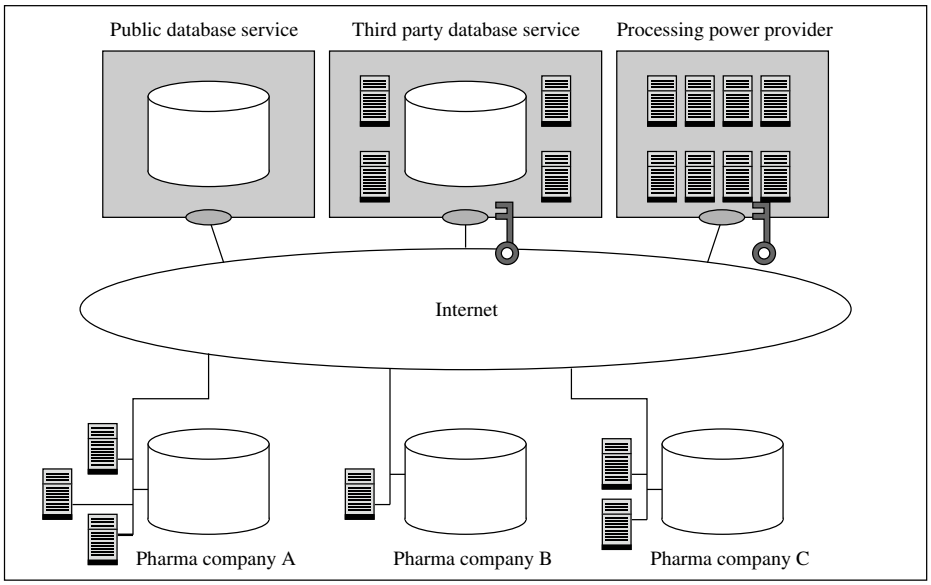


Fig. 2. Improved Service-Oriented Architecture

functionality to be used in a distributed system. It is responsible for distributing queries across multiple databases, which can be decomposed into several steps.

In a simplified form of the steps presented by Özsu et al. [11], the distributed execution of queries involves the splitting of a query across the databases, the translation of queries into the respective languages for each database and the integration of results.

The access to the databases in our example could e.g. be exported as an OGSA-DAI [10] service. OGSA-DQP, an engine for distributed query processing [1], can perform distributed queries over data sources defined with OGSA-DAI.

4 Architectural Considerations

We have outlined how the system is currently structured and employed in section 2. Along with that, we have pointed out the major problem with this system. A natural way to improve it, reducing the occuring problems, is to cast database access and integration, and processing into *services*. The architecture of this improved, Grid-like system is shown in figure 2.

In the remainder of this section, we discuss the advantages, disadvantages, and consequences of restructuring different parts of the system as services.

4.1 Public Databases

By offering access to the public databases as a Grid service, the need to repeatedly download, replicate, and integrate the data is cancelled. Database maintenance then only has to be performed by the database host. The consumption of bandwidth is reduced since with the service-based approach, only the required data is queried from the databases.

4.2 Third Party Contractors

The access to data provided by third party contractors can also be done via services. Since these data are not public and only to be accessed by pharma companies paying for it, AAA mechanisms (authentication, authorization, and accounting) are required to restrict access.

4.3 Public and Third Party Databases

The transformation of data access into a service solves problems, but also introduces new ones. In section 2, we have stated that we do not want outsiders to deduce knowledge about currently running projects from the request pattern of the pharma companies. With the current architecture, this problem does not occur since little knowledge is gained from the fact that a full database is used by a pharma company. Using the service-based approach, the data providers get to see the request patterns of the pharma companies, yielding more comprehensive knowledge.

In order to achieve the goal, the actual requests have to be disclosed from the data providers as far as possible. To this end, bogus requests might have to be interspersed in the stream of requests originating from a pharma company. This of course has to be done before the stream of requests reaches the data provider's database.

Another advantage of providing data access via a service is that new databases or databases other than the purchased ones can be automatically discovered. If the data services are semantically annotated, pharma companies can learn of databases which might be useful to them, but they have not purchased yet.

4.4 Processing Power Providers

The Grid approach also gives rise to new business opportunities. E.g., in figure 2, a provider of processing power was added to the system. Since data integration does not necessarily have to be performed at the respective pharma companies any more, data processing could be outsourced by them. By seeing these processing providers as contracted partners, we assume their machines to be trusted parties. Since potentially all pharma companies participating in the system might want to outsource their processing resources, again AAA mechanisms are required to restrict access to the nodes installed by the processing providers. For the strict security and confidentiality reasons mentioned above, processing

jobs originating from different pharma companies must be completely shielded from each other. Neither must a pharma company A see what kind of processing is performed by company B, nor must it see any of its data involved in the processing.

4.5 Distributed Querying

In order to execute queries over the distributed databases, we plan to use the toolkits mentioned in section 3. By exporting database access as an OGSA-DAI service and using OGSA-DAP for distributed querying, we can use existing standards and implementations to get a working system soon. Then, we plan to transparently improve the distributed querying facility by employing a P2P overlay approach.

As shown [1] and in a simplified version in figure 3, distributed querying requires *query planning*. A query plan is the result of compiling and optimizing a query. As indicated by the dashed lines, this plan is partitioned across the databases, which execute their parts of the query. The partial results have to be integrated afterwards, i.e. by the operation depicted by the node in the overlapping region of the partitions.

There are two degrees of freedom in this: First, the partitioning of the query plan. This might require redundant data storage at multiple sites. And second, the implementation of the physical operators for integrating the results (typcially join operations). PIER is an example of how join operations can be implemented using a DHT [7]. Both degrees of freedom require the definition of a cost model for making appropriate choices.

E.g., if the databases are arranged in a P2P overlay, the link weights (delay, bandwidth) could be used to guide the selection of particular partitioning. The integration of the partial results does not necessarily have to take place at one of the database sites. We assume that each pharma company defines a subset of all nodes in the VO as trustworthy, i.e. by using AAA mechanisms. Then, the task of integrating partial results can be assigned to any node in this trustworthy node subset.

This approach of shipping queries together with partial results to nodes also has to take into account the transmission and processing costs of jobs. We envision a P2P overlay formed by the set of trustworthy nodes which organizes itself according to a cost model for query shipping. Queries can then be routed in this network towards suitable nodes, providing a load balancing mechanism. By dynamically adding more nodes to the query overlay, e.g. by renting more processing power from a provider, the system can be scaled in a self-organizing way.

5 Future Work

The next steps in our project involve a detailed analysis of the available database types and typical queries posed to the system. With this information, we can

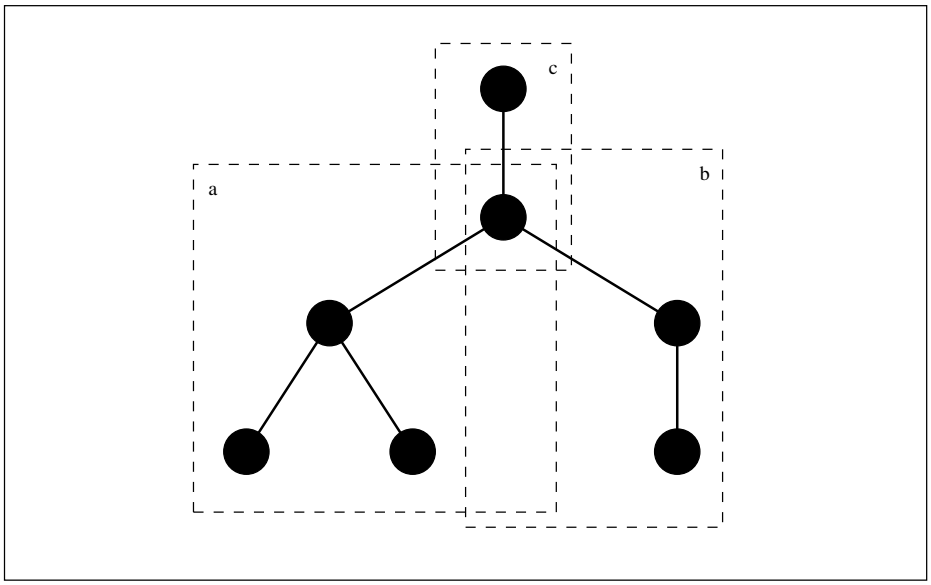


Fig. 3. Query Plan for Distribution

further specify the requirements of the distributed query processing component. Important topics there are the formation of the query overlay, the implementation of physical database operators using P2P approaches, and suitable AAA mechanisms and policies for achieving the rigid security requirements of the application.

References

1. M. Nedim Alpdemir, Arijit Mukherjee, Norman W. Paton, Paul Watson, Alvaro A. A. Fernandes, Anastasios Gounaris, and Jim Smith. Service-Based Distributed Querying on the Grid. In *Proceedings of the International Conference on Service-Oriented Computing*, 2003.
2. Fran Berman, Geoffrey Fox, and Tony Hey, editors. *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Ltd, Wiley Series in Communications Networking & Distributed Systems, The Atrium, Southern Gate, Chichester, West Sussex PO19 8SQ, England, 2003.
3. Ian Foster and Adriana Iamnitchi. On Death, Taxes, and the Convergence of Peer-to-Peer and Grid Computing. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Berkeley, CA, USA, February 2003.
4. Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. *The physiology of the Grid*, pages 217–249. In Berman et al. [2], 2003.
5. Ian Foster, Carl Kesselman, and Steven Tuecke. *The anatomy of the Grid*, pages 171–197. In Berman et al. [2], 2003.

6. K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of DHT routing geometry on resilience and proximity. In *Proceedings of the SIGCOMM 2003 conference*, pages 381–394. ACM Press, 2003.
7. Ryan Huebsch, Joseph M. Hellerstein, Nick Lanham, Boon Thau Loo, Scott Shenker, and Ion Stoica. Querying the Internet with PIER. In *Proceedings of the 29th International Conference on Very Large Databases (VLDB)*, Berlin, September 2003.
8. Gene Kan. Gnutella. In Andy Oram, editor, *Peer-to-Peer. Harnessing the Power of Disruptive Technologies*, pages 94–122. O'Reilly, Sebastopol, CA, 2001.
9. Peter Z. Kunszt and Leanne P. Guy. *The Open Grid Services Architecture, and Data Grids*, pages 385–407. In Berman et al. [2], 2003.
10. Miscellaneous authors. Open Grid Services Architecture Data Access and Integration (OGSA-DAI), 2004. http://www.ogsa-dai.org.uk, accessed on 01 June 2004.
11. M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems*. Prentice-Hall, Englewood Cliffs, New Jersey 07632, USA, 1991.
12. Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the SIGCOMM 2001 conference*, pages 149–160. ACM Press, 2001.
13. Paul Watson. *Databases and the Grid*, pages 363–384. In Berman et al. [2], 2003.

Distribution Alternatives for Superimposed Information Services in Digital Libraries

Sudarshan Murthy[‡*], David Maier[*], Lois Delcambre[*]

Department of Computer Science, OGI School of Science and Engineering at OHSU
20000 NW Walker Road Beaverton, OR 97006 USA
{smurthy, maier, lmd}@cse.ogi.edu
http://www.cse.ogi.edu/sparce

Abstract. Component-based service-oriented digital library (DL) architectures are being used to provide superimposed information services such as annotations. Although much attention is paid to the issues in building components for these services, not enough attention has been paid to their deployment, specifically to distribution. We believe that extending DL services with superimposed information services is important, and that it is essential to understand the distribution alternatives for the components that provide such services. We use our middleware architecture for superimposed information management, called the *Superimposed Pluggable Architecture for Contexts and Excerpts (SPARCE)*, for illustration. We describe four distribution alternatives and their trade-offs. We also define some metrics to compare the performance of the alternatives.

1 Introduction

Our research on superimposed information focuses on allowing users to *superimpose* new information such as annotations and summaries on top of existing *base* information such as web pages and PDF documents. In addition to superimposing annotations, a user may select parts of existing information and create new linkages among those selections. For example, a user may create an alternative organization of sections in a PDF document, or the user may create a table of selected contents. We use a component-based middleware architecture called the *Superimposed Pluggable Architecture for Contexts and Excerpts (SPARCE)*, [2], for superimposed information management.

Some digital Library (DL) systems too support creation of annotations and metadata over their information [9, 7] using component-based architectures. Some of the benefits of component-based architectures are that architectural components may be replaced with alternatives and new components may be plugged in easily. They make it easier to build new services using component stacks.

Another benefit of component-based architectures, one that does not receive as much attention, is flexibility of deployment. With proper interface design and abstraction, components (both data and executable) may be deployed centrally or distributed, without affecting the services they provide. This flexibility is important

[‡]Author's work has been supported by US NSF grant IIS 0086002.
[*]Author's work has been supported by US NSF grant IIS 9817492.

because placing a component at the right location can improve performance, especially for frequently used services.

In this paper, we present four distribution alternatives and their trade-offs when providing superimposed information services such as annotations in a DL. We use our component-based middleware architecture called the *Superimposed Pluggable Architecture for Contexts and Excerpts (SPARCE)*, [2], to illustrate the alternatives. We present the alternatives without a specific DL architecture in mind because they should apply to component-based DL architectures in general.

To motivate, we present a simple ODL [5] annotation system (Figure 1) and describe two distribution alternatives for it. This system has three components: user interface, annotation, and an archive. A distribution alternative, Alternative A, is to run the user interface and the annotation components on a patron's (client's) computer, and run the archive component within a DL server. Alternative B is to run the user interface on a patron's computer, and run the other two components within a DL server. These alternatives could differ significantly in performance and maintainability. For example, Alternative B may be more maintainable because few components run outside the DL server, but it has the potential to increase the load on the server. The alternatives differ also in the interface a DL server needs to provide to the outside world. Alternative A requires a DL server to provide interface to the Archive component (the Annotate component connects from the patron's computer to the Archive component in the DL server), whereas Alternative B requires the server to provide interface to the Annotate component.

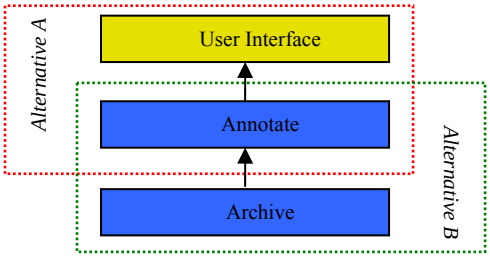


Fig. 1. An ODL-style annotation system. Each dotted rectangle contains components collocated in the distribution alternative called out

Proximity of components can affect the overall system performance. To give an idea, we present results from a simple experiment we conducted. Table 1 shows the mean round-trip time for three web-service methods based on SOAP [8] bound to HTTP. From the table we see that the mean round-trip time over a WAN is about 60 times that over a LAN when sending and receiving 400 bytes. Although the exact numbers would depend on the method-invocation technology employed, comparable ratios are common with any technology.

The numbers in Table 1 tell us that performance could be improved by placing components with a higher number of round trips between them closer to each other (based on network distance, not geographic distance). For example, if we know that the Annotate component makes many round trips to the Archive component to serve a

single request from the user interface, we may benefit by collocating the Annotate and Archive components in a DL server as in Alternative B.

The rest of this paper is organized as follows. Sections 2 and 3 give an overview of superimposed information and SPARCE respectively. Section 4 defines some metrics, and details four distribution alternatives and their trade-offs. Section 5 discusses some issues in employing the distribution alternatives. Section 6 provides a brief overview of related work. Section 7 concludes the paper.

Table 1. Mean round-trip time (in milliseconds) for SOAP-based web service methods via HTTP. Columns *Input* and *Output* denote the number and type of inputs and outputs respectively for the methods; Column *Local* shows round-trip time when client and server are on the same computer, *LAN* shows round-trip time when client and server are connected over a LAN (100 MBPS, one hop), *WAN* shows round-trip time when the client and server are connected over a WAN (756 KBPS DSL connection, more than 18 hops)

		Mean round-trip time (milliseconds)		
Input	Output	Local	LAN	WAN
None	None	2.94	3.13	146.74
10 integers	10 integers	3.13	3.36	137.53
One 400-byte array	One 400-byte array	7.87	10.53	689.39

2 Superimposed Information

Superimposed information refers to data placed over existing information sources to help select, access, organize, connect, and reuse information elements in those sources. Existing information sources reside in the *base layer*, and data placed over one or more base sources resides in the *superimposed layer* (see Figure 2(a)). Word-processor documents, databases, and web pages are examples of base documents. A stand-off annotation is an example of superimposed information (because it is stored separately from the object of annotation, and it maintains a link to the object). An application that manipulates base information is called a *base application*; an application that manipulates superimposed information is called a *superimposed application*.

2.1 Marks

A superimposed information element (such as an annotation) refers to a base information element (such as a selection in a spreadsheet) using an abstraction called a *mark*. Figure 2(a) shows the superimposed layer using marks to address base elements. Several implementations of the mark abstraction exist, typically one per base type. In order to address a selection in a base document, a mark implementation supports an addressing scheme appropriate for that base type. For example, an implementation for PDF documents may use page number and index of the first and last words in a text selection, whereas an implementation for XML documents may use XPath. All mark implementations provide a common interface to address base information, regardless of the base types or access protocols they support. A superimposed application can work uniformly with any base type by virtue of this common interface.

Figure 2(b) shows some superimposed information elements created and organized in our superimposed application called *RIDPad* [2]. It shows four items labeled 'Statement', 'FONSI', 'Details' and 'Issues,' each linked to a selection in the base layer. For example, the item labeled 'Issues' is linked to a selection in an MS Excel spreadsheet; the item labeled 'FONSI' is linked to a selection in a MS Word document. The box labeled 'Decision' groups items. Figure 3(a) shows the MS Word mark of the 'FONSI' item activated.

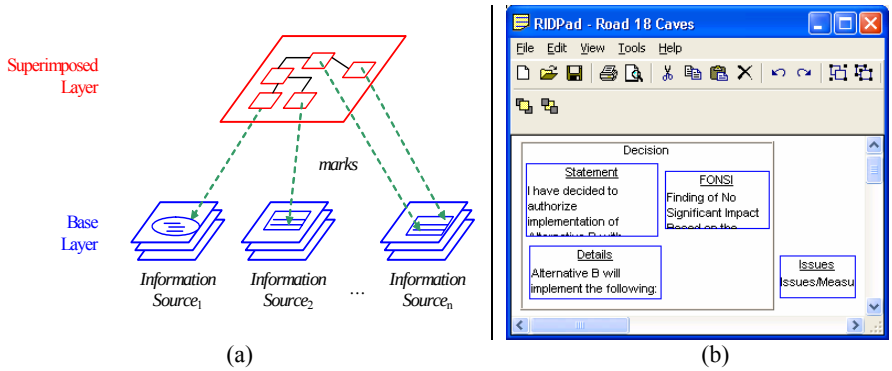
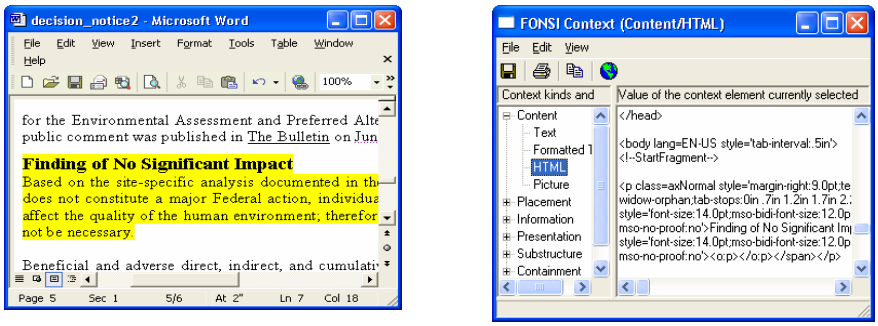


Fig. 2. (a) Layers of information in a superimposed information management system. A *mark* connects a superimposed information element to the base layer. (b) Superimposed information organized using *RIDPad*, a superimposed application

2.2 Excerpts and Contexts

Superimposed applications may occasionally need to incorporate the content of base-layer elements in the superimposed layer. For example, an application might use the extracted base-layer content as the label of a superimposed element. We call the contents of a base-layer element an *excerpt*. An excerpt can be of various types. For example, it may be text or an image. An excerpt of one type might also be transformed into other types. For example, formatted text in a word processor could also be seen as plain text, or as a graphical image.

In addition to excerpts, superimposed applications may use other information related to base-layer elements. For example, an application may group superimposed information by the section in which the base-layer elements reside. To do so, the application needs to retrieve the section heading (assuming one exists) of each base-layer element. We call information concerning a base-layer element, retrieved from the base layer, its *context*. Presentation information such as font name and location information such as line number might be included in the context of a mark. Each such piece of information is a *context element*, and context is a collection of context elements. Because we use the same mechanism to support both contexts and excerpts, we often use the term "context" broadly to refer to both kinds of information about a base-layer element. Figure 3(b) shows the context of a MS Word mark in a browser (for the mark activated in Figure 3(a)).



(a) (b)

Fig. 3. (a) An MS Word mark activated. The highlighted area is the marked region. (b) The context of the MS Word mark in the Context Browser. The browser is showing part of the HTML markup required to display the excerpt for the mark formatted as is in the base layer

3 SPARCE

The *Superimposed Pluggable Architecture for Contexts and Excerpts* (*SPARCE*) is a middleware-based approach for mark and context management [2]. It is designed to be extensible in terms of supporting new base-layer types and context-element types, without adversely affecting existing superimposed applications. Figure 4 shows the SPARCE reference model. The Mark Manager provides operations such as creating and storing marks. It also maintains a marks repository. The Context Manager retrieves context information. Superimposed applications use the managers to create marks and access context (which in turn use base applications).

The Mark Manager supports three operations for marks: creation, retrieval, and activation. *Mark creation* is the operation of generating a new mark corresponding to a selection in a base layer. This operation consists of three steps: generating the address of base information (and other auxiliary information), using the information generated to create a mark object, and storing the mark object in the mark repository. Details of each mark, such as the address of the container and the selection inside it, are stored as an XML file. The mark repository is a database of such XML files.

The *mark retrieval* operation returns a mark from the mark repository. *Mark activation* is the operation of navigating to a location inside the base layer, using the information supplied by a mark.

SPARCE uses mediators called *context agents* to retrieve context information for a mark from the base layer. A context agent interacts with a base application to retrieve context. The name of the context agent to use for each mark instance is one of the details stored in the mark repository. SPARCE uses this information to instantiate an appropriate context agent for a mark instance. A superimposed application receives a reference to the instance of context agent from SPARCE, and then works directly with the agent instance to retrieve context.

The components of SPARCE (see Figure 4) may be mapped to those of the ODL-style annotation system we introduced in Figure 1. Superimposed applications (patron applications) provide the user interface. These could be desktop applications or browser-based applications (applets) running on a patron's computer. The Mark

Manager, the Context Manager, and the base applications constitute the Annotate component. The base documents and a DL's interface to access them (if required) roughly constitute the Archive component.

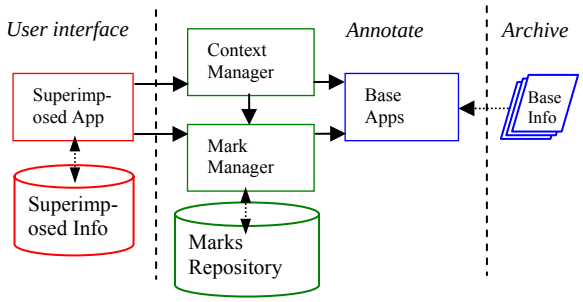


Fig. 4. The SPARCE reference model. Solid arrows show dependency, dotted arrows show data flow (not all data-flows shown). The dashed lines map components to those in Figure 1

4 Distribution Alternatives

We describe four distribution alternatives for SPARCE (see Figure 6) when providing superimposed information services in a DL.

Four components of SPARCE (see Figure 4) are candidates for distribution: the patron (superimposed) application, the Mark Manager, the Context Manager, and base applications. For simplicity, we assume the following for all alternatives:

- The DL server contains base documents (and that an appropriate interface in the DL server is used to access the documents).
- The two manager modules (the Mark Manager and the Context Manager) run on the same computer.
- The mark repository is stored wherever the Mark Manager is deployed.
- The patron application always runs on a patron's computer.
- The superimposed information is stored on a patron's computer.

We first present a goal for distribution and some related metrics. We do not provide results based on these metrics, but estimate trends based on a few rules of thumb (see Section 4.6). We assume a patron uses a high-speed (such as broadband) Internet connection to a DL server. We also assume that the ratios of the mean round-trip times shown in the third row (for 400-byte array input and output) of Table 1 hold.

A note on terminology: the term "patron's computer" in the rest of this paper may mean a stand-alone computer or a computer in a network local to the patron. Our description of alternatives would be valid for either meaning of the term.

4.1 Goals and Metrics

Several metrics such as latency (for example, time to serve a request), load (for example, the number of active processes) and throughput (for example, the number of requests processed per unit time) should be considered for a thorough analysis of the alternatives. However, due to space constraints, we discuss only latency.

The latency of a patron application's request (T_{pm}) is the duration between the patron initiating a request in the patron application (to a manger module) and the patron receiving a corresponding response. It is made up of the following components (see Figure 5):

- t_{bb}: Time taken by a base application to complete a requested operation. For example, time to retrieve a context element's value.
- t_{mb}: Round-trip time between a manager module and a base application. This time measures the duration between a manager module receiving a request from a patron application and the manger module returning a corresponding response, after discounting the time the base application takes to complete its work.
- t_{pm}: Round-trip time between a patron application and a manager module. This time measures the duration between the patron initiating a request in the patron application (to a manger module) and the patron receiving a corresponding response, after discounting the time the manager module needs to complete its work.

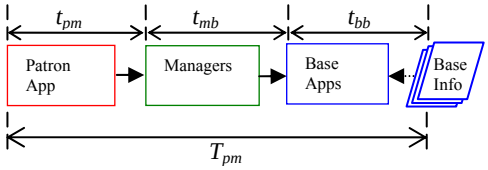


Fig. 5. Components of latency of a patron application's request. The subscripts *p*, *m*, and *b* stand for patron application, manager module, and base application respectively. They identify the pair of architectural components with which a latency term is associated

In distributing the architectural components, one of our goals is to minimize the latency of a patron application's request (T_{pm}). Because this latency is the sum of the times t_{pm}, t_{mb}, and t_{bb}, our sub-goals are to minimize these terms. The distribution alternatives we describe vary the location of architectural components to highlight the affect of each alternative on these latency terms.

4.2 Distribution Alternative A

Alternative A is to run the patron applications, the manager modules, and the base applications on a patron's computer (Figure 6). That is, a DL server only supplies base information. The patron opens base documents using appropriate base applications running on his or her computer. Marks and superimposed information are stored on the patron's computer. Because all components run within the patron's computer, the round-trip times between them would be quite low. Base documents would still be accessed over the network because they reside within the DL server. This cost is likely incurred only once per document per session, because documents accessed over the Internet are usually (automatically) first downloaded to the client's computer. Base applications then access the documents locally. This alternative requires that each patron have all base applications locally, even for rarely encountered base types.

4.3 Distribution Alternative B

Alternative B is to run the patron applications on a patron's computer, and run the manager modules and the base applications within a DL server. Marks are stored in the DL server, and superimposed information is stored on the patron's computer. Because the base applications operate within the DL server, the patron would be able to view base documents in their native applications only if those applications are also available locally on the patron's computer. However, because the manager modules run inside the DL server, the mark activation operation would be unable to exploit any base application available on the patron's computer. When the patron activates a mark (for example, the MS Word mark in Figure 3(a)), the DL server prepares the context elements needed to display the excerpt and sends it to the patron application (possibly in HTML as in Figure 3(b)). The patron may request additional context elements to view as needed.

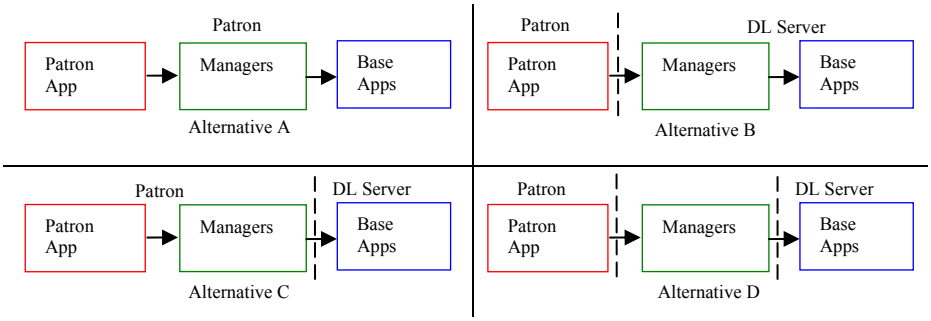


Fig. 6. Distribution alternatives for SPARCE. The dashed lines denote network boundaries

Because the manager modules and the base applications run within a DL server, the round-trip time between them would be low, but the round-trip time between a patron application and the manager modules (t_{pm}) would be high. Consequently, patron applications must strive to minimize the number of round-trips to the manager modules. Combining requests for context elements can reduce the number of round trips.

4.4 Distribution Alternative C

Alternative C is to run the patron applications and the two manager modules on a patron's computer, but run the base applications within a DL server. As in Alternative A, marks and superimposed information are stored on the patron's computer. Like Alternative B, the patron would be able to view base documents in their native applications only if those applications are also available locally. Unlike Alternative B, because the manager modules run on the patron's computer, the mark activation operation would be able to exploit any base application available on the patron's computer. In other cases, the Context Manager module would have to retrieve the necessary context elements to provide a view of marked regions. Because the manager modules run on the patron's computer, but the base applications run within the DL server, the round-trip time between them (t_{mb}) would be high.

4.5 Distribution Alternative D

Alternative D is to run the patron applications modules on a patron's computer, the two manager modules in a *middle tier*, and run the base applications within a DL server. As in Alternative A, the superimposed information is stored on the patron's computer, but marks are stored in the middle tier. The capabilities of a patron application are similar to those in Alternative B. The performance of this alternative is also similar to that in Alternative B, except the round-trip time t_{mb} would also be higher because the manager modules run in the middle tier.

With the manager modules running in a middle tier, they can connect to more than one DL server. Further, it may be possible to deploy the base applications in the middle tier to minimize the round-trip time between the manager modules and the base applications. A federation of DLs may like to maintain such a middle tier to distribute the cost of operations.

4.6 Summary of Distribution Alternatives

Table 2 provides a summary of the location of components, the role of the DL server, and the profile of the components on the patron's computer for each alternative. A DL server operating as an *information server* is similar to a file server, whereas an *application server* runs applications on behalf on clients. A *thin* client profile means a minimal amount of code runs on the patron's computer. Patron applications tend to be browser-based (applets for example). A *fat* client profile means large amounts of code run on the patron's computer. Patron applications tend to be desktop applications, and are often richer in functionality than browser-based applications.

Table 2. Summary of alternatives. *Client profile* is the profile of the components on the patron's computer

	Location of components				
Alternative	Patron apps	Managers	Base apps	DL Server Role	Client profile
A	Patron	Patron	Patron	Info server	Fat
B	Patron	DL	DL	App server	Thin
C	Patron	Patron	DL	App server	Fat
D	Patron	Middle tier	DL	App server	Thin

Table 3 summarizes the *trend* we expect for maintenance cost and performance of the resulting systems from the alternatives. Two rules of thumb drive our expectation of maintenance cost: A thin client is less expensive to maintain than a fat client, and it is less expensive to maintain components that run within a DL server than those deployed elsewhere. For components that run outside the DL server, changes made to a component need to be propagated to all locations where that component is deployed.

The load on a DL server increases as the number of components running within the server increases. The trend *Medium* for Alternatives C and D indicates that the load on the DL server would be greater than that for Alternative A, but less than that for Alternative B. (The manager modules run outside the DL server in Alternatives C and D.)

Two rules of thumb guide our expectation of round-trip times: placing components closer to each other reduces the round-trip time between them; the reduction is greater

if the number of round-trips between them is large (especially when the components exchange large amounts of data).

Table 3. Summary of maintenance cost and performance. Columns t_{pm} and t_{mb} are round-trip times as defined in Section 4.1

	Maintenance Cost				Round-trip time	
Alternative	Patron apps	Managers	Base apps	DL Server Load	t_{pm}	t_{mb}
A	High	High	High	Low	Low	Low
B	Low	Low	Low	High	High	Low
C	High	High	Low	Medium	Low	High
D	Low	Low	Low	Medium	High	High

5 Discussion

In reality, DL systems are likely to employ a mixture of distribution alternatives. For example, some DL providers might wish to support patron applications of different capabilities (for example, fat clients and thin clients). Doing so requires the DL server to provide many kinds of interfaces (expose manager modules *and* base applications) for patron applications to choose from. Also, a patron may work with more than one DL, and those DLs may employ different distribution alternatives. In this case, patron applications must be able to discover the alternative a DL system employs.

Hosting base applications outside a patron's computer (as in Alternatives B, C, and D) can cause some problems. Without the necessary base application available locally, a patron will be unable to see a mark in its context (as in Figure 3(a)). In such cases, the context manager would have to retrieve the context elements necessary to provide a "broad enough" view of the selection, but the combined size of the context elements could be excessively large. Alternatively, the context manager could retrieve the context elements needed to display just the excerpt of the mark (but nothing surrounding it). In either case, the context manager needs to transform context elements to a format such as HTML or GIF, so the patron application may render the view. This transformation may not be easy for some base types. This problem is more likely with Alternatives B and D, because the manager modules run outside the patron's computer, and they are unable to "call back" base applications on a patron's computer. Mechanisms do exist for manager modules to call back applications on patron's computers, but they present security concerns. Also, calling back may require the manager modules to cope with many versions of the same base application.

Sharing annotations is an emerging need among DL patrons. When sharing superimposed information, the corresponding marks may be shared or replicated. It is also possible to share just the marks, but not the superimposed information that use them. In reality, we envision that some marks and superimposed information may be shared, and some marks may be replicated.

Distributing components and sharing information each increase security risks. DL systems may need to implement more than one alternative to balance security and performance. A small number of DL server interface points, and narrow functionality of those interface points can help reduce security risks. For example, the number of interface points in the SPARCE manager modules is fewer than that in most base

applications. They are also narrower in functionality. If a patron connects to a DL server over a public network such as the Internet, it may be better to present an interface to the manager modules rather than to base applications (Alternatives B and D). However, base applications may be exposed to a patron connecting over a local intranet (Alternative C).

In discussing the distribution alternatives, we mentioned that combining requests to retrieve contexts can help reduce latency. Such intelligence may be added to the Context Manager, thus benefiting all patron applications. Caching context may also be useful for some applications. The location of the cached context may be chosen based on needs. A context cache placed within a DL server can help with requests from many patrons, whereas a cache on a patron's computer can help with requests from only that patron. Finally, a DL server may replicate instances of the manager modules and the base applications to handle large number of requests.

6 Related Work

The DELOS-NSF Working Group's report on Digital Library Information-Technology Infrastructures [1] highlights the importance of services and infrastructure for metadata and annotation services in DLs. OAI-PMH and its extension XOAI-PMH have demonstrated the feasibility of component-based architectures for metadata and annotation services respectively in a DL [7, 6].

Some DL systems that support annotations do not consider distribution alternatives of both data and executables. The UC Berkeley Digital Library Project uses *superimposed behaviors* in multivalent documents to support annotations [9]. That work facilitates distributed annotation and base data, but not distribution of executables. InfoBus [3] defines a mechanism for interaction among UI clients, proxies, and repositories. (SPARCE's manager modules may be viewed as proxies.) It also defines service layers that are available to clients, proxies, and repositories. However, it does not consider distribution of executables.

FEDORA [4] and XOAI-PMH [6] provide promising frameworks for integration of superimposed information services with other DL services. The *parameterized disseminators* of FEDORA could be used to address (access) parts of documents. XOAI-PMH does not explicitly specify sub-document objects, but its extensibility mechanism could be used a create *item* instances that serve the purpose.

7 Summary

We have described four distribution alternatives to provide superimposed information services in DLs and defined some metrics to compare the performance of the alternatives. We have illustrated the distribution alternatives using SPARCE, our middleware architecture for superimposed information management. We have also discussed some of the issues in employing the distribution alternatives.

References

1. DELOS-NSF Working Group on Digital Library Information-Technology Infrastructures: Report (2002). Available online: http://www-rocq.inria.fr/~abitebou/pub/DELOS-ITI.pdf

2. Murthy, S., Maier, D., Delcambre, L., Bowers, S.: Putting Integrated Information in Context: Superimposing Conceptual Models with SPARCE. In: Proceedings of the First Asia-Pacific Conference of Conceptual Modeling, Dunedin, New Zealand (2004) 71-80
3. Roscheisen, M., Baldonado, M., Chang, C., Gravano, L., Ketchpel, S., Paepcke, A.: The Stanford InfoBus and Its Service Layers: Augmenting the Internet with Higher-Level Information Management Protocols. Digital Libraries in Computer Science: The MeDoc Approach. Lecture Notes in Computer Science, Vol. 1392, Springer (1998)
4. Staples, T., Wayland, R., Payette S.: The Fedora Project: An Open-source Digital Object Repository Management System. D-Lib Magazine. Vol. 9, Number 4 (2003)
5. Suleman, H., Fox, E.A.: A Framework for Building Open Digital Libraries. D-Lib Magazine. Vol. 7, Number 12 (2001)
6. Suleman, H., Fox, E.A.: Designing Protocols in Support of Digital Library Componentization. In Proceedings of ECDL 2002. Rome, Italy (2002)
7. The Open Archives Initiative: Protocol for Metadata Harvesting, Version 2.0. (2002).
8. W3C XML Protocol Working Group: Simple Object Access Protocol. (2003)
9. Wilensky, R.: Digital library resources as a basis for collaborative work. Journal of the American Society of Information Science. Vol. 51, Number 3 (2000) 228-245

JDAN: a Component Architecture for Digital Libraries

Fabio De Rosa[1,2], Alessio Malizia[2], Massimo Mecella[1]
Tiziana Catarci[1], and Luigi Cinque[2]

[1] Università di Roma "La Sapienza"
Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
Via Salaria 113 (2nd floor), 00198 Roma, Italy
{derosa,mecella,catarci}@dis.uniroma1.it
[2] Università di Roma "La Sapienza", Dipartimento di Informatica
Via Salaria 113 (3rd floor), 00198 Roma, Italy
{malizia,cinque}@di.uniroma1.it

Abstract. The process of converting documents in digital forms is fundamental for office automation, and is continually becoming a more and more powerful tool in those fields where information still comes from hybrid sources (papers, books, files, etc), e.g., in digital libraries.
In this paper we present the JDAN (Java-based environment for Document Applications on Networks) framework, that, on the basis of a component architecture, is able to manage both document images and forms. We argue that JDAN could be a starting point for developing more complex architectures for digital libraries, as it is based on XML technologies and has a good modularization that allows its integration in both Service- and Grid-based scenarios.

1 Introduction

The process of converting documents in digital forms is fundamental for office automation, and is continually becoming a more and more powerful tool in those fields where information still comes form hybrid sources (papers, books, files, etc).

Up to now, many different approaches have been used but widely agreed upon standards are still lacking. Typical problems of document analysis systems are: layout segmentation and syntactic parsing, but also the selective extraction of information such as document types and semantic contents; most document processing packages are designed either for document recognition (i.e., indexing and archiving of document images) or for data acquisition (i.e., extracting data from filled forms).

Document recognition is essentially the process of converting paper documents into digital images and indexing such data. Images are stored as data files (typically as TIFF files) and together with indexes are stored in a content management system. As far as data acquisition is concerned, most form-oriented

products available on the market today instead require the user to redesign his forms in order to achieve acceptable recognition rates.

In this paper we present the JDAN (Java-based environment for Document Applications on Networks) framework, that, on the basis of a component architecture, is able to manage both document images and forms. We argue that JDAN could be a starting point for developing more complex architectures for digital libraries, as it is based on XML technologies and has a good modularization that allows its integration in both Service- and Grid-based scenarios. Section 2 describes the architecture of the proposed framework, whereas Section 3 presents the details of the different components of JDAN. Finally Section 4 considers the deployment of the proposed framework and concludes the paper with some final remarks.

2 The JDAN Framework and Architecture

The proposed framework offers different functionalities:

- **Data Acquisition** for handling scanning; it could be deployed on a specific center where high-speed expensive equipments are used to acquire multiple format documents. The framework supports standard definitions that allow changing both the indexing and recognition rules even after the data acquisition phase. Indeed, many available systems use batch processes to enhance performances and reduce costs, but they are wrapped around the capture process. If new requirements rise (e.g., update of the forms), the batch process needs to be halted and the documents to be scanned again in compliance with the new specifications [3]. Conversely with the segmentation module proposed in JDAN, users can acquire batches and batches of documents only specifying the zone type, size and label, then the recognition module automatically detects where the zone is on the document, thus resulting in a reduction of costs. Section 3.1 provides more details on this issue.
- **Recognition**, whose peculiarity is the possibility of introducing different recognition modules, on the basis of market standards or research results. The aim of JDAN was to make feasible to plug-in and substitute the recognition module depending on the type of documents; users adopting the framework can develop their own modules on top of the JDAN specification and also include OCR modules for the regions obtained from the defined segmentation, thus reducing the manual annotation in document forms.

[3] As an example, if the organization has a batch of 3000 paper documents and while scanning the first 2000 it finds that something in the forms has changed (e.g., the position of the name and surname labels), with most of the currently available applications the organization has to stop the process, change the area specifications from which the information will be captured and then start again, thus impacting the whole acquisition process in terms of hours.

- **Indexing** could be automatic or manual, depending on a set of parameters that could be tuned after evaluating the segmentation and classification results. By using such parameters the system can decide which kind of indexing phase to perform: either manual or automatic. Manual indexing requires users to enter information from forms, even if by adopting automatic recognition only those documents out of the parameters range will be presented to the user for manual data-entry. Moreover by using both automatic and manual segmentation we could perform semi-automatic indexing, which could help in validation for manual data-entry. In order to avoid typing error, the automatic indexing is performed and then the results are presented for manual annotation, and if differences appear the document has to be indexed again.
- **Storing** of indexes and original document images into SQL standard databases. The engine of the framework is based on a standard J2EE container that uses an XML-based index definition schema. XML index descriptions reside only in the central repository while images could be spread across multiple sites, thus saving costs in terms of need of central mass storage and broadband expensive connections, as document images remain within organizations while only indexes are centralized.
- **Querying** will be performed using clients with XML-parsing features. The framework supports both thin clients based on browsers accessing servlets and JSPs via HTTP, and fat clients accessing via RMI/IIOP.

Table 1. JDAN architectural goals and adopted technologies

Feature	**Goals**	**Technologies**
segmentation and classification	automatic image cleaning, region recognition and classification: text, image and graph	gcc compiler, XML, OCR modules
indexing	local and remote indexing, automatic region indexing, internet-based indexing	Apache web server, servlets and JSPs, JDBC, MySQL (potentially any other SQL-based DBMS)
scalability and reliability	expanding the system according to throughput requirements and security data exchange	JBOSS application server, MySQL, HTTPS support
integration	importing segmentation and classification algorithms, choosing data format to export	XML

Table 1 outlines the main features of the JDAN framework and the set of technologies that have been adopted in order to reach specific goals. As much as possible the design choices have been to adopt open and standard technologies,

and, as further detailed, the integration of the different components is gained through XML.

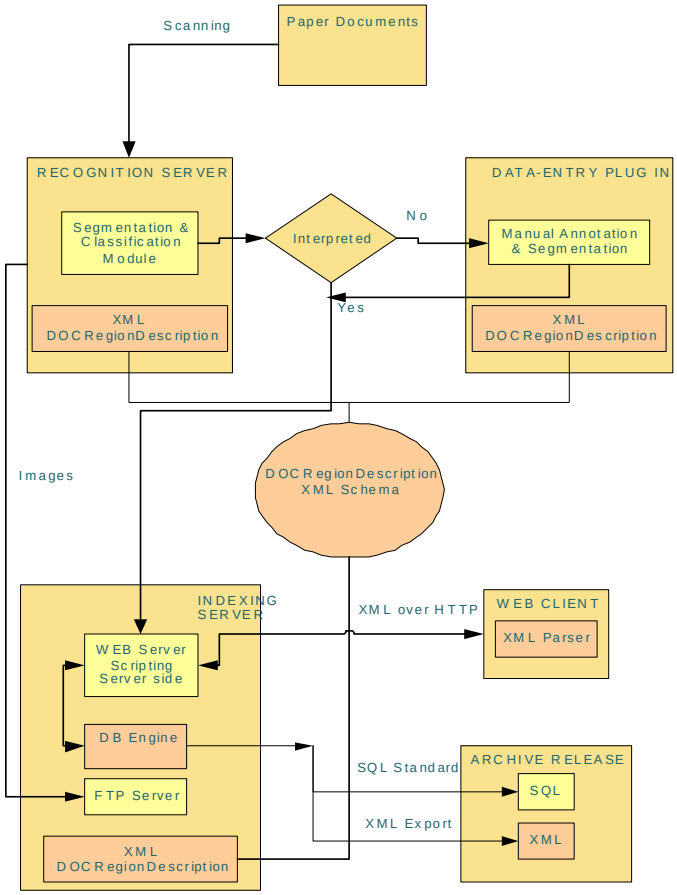


Fig. 1. The JDAN framework

In Figure 1 the four main components of the framework are shown: the *Recognition Server*, the *Indexing Server*, the *Data-entry Plug-in* (for annotation), and the *Web Client* (i.e., the query module). Moreover we can define also an *Archive Release* module if we want to export our indexes and images to other applications.

The core of the JDAN environment is the `DOCRegionDescription` XML Schema, which is a published description for the result of a segmentation process. This approach increases interoperability of different software packages, thus allowing cooperative scenarios in order to solve document recognition related problems (document analysis, image processing, segmentation, etc.): developers

of different software packages write their own `DOCRegionDescription`-compliant XML specification and their applications can be plugged on top of JDAN [4].

Indeed one of the most time-consuming challenges for developers has been to exchange data between varieties of different systems. Converting data to XML can greatly reduce this complexity, thus creating indexes that can be read by many different types of applications. XML can also be used to store data in files or in databases. We have defined a standard description for the result of a segmentation process, in order to let users choosing which algorithm best fits their needs. Therefore our framework could be used in a research organization as well as in a business department: students and researchers that already have developed their segmentation algorithms could use the environment to test them; while in a business department state-of-art segmentation software could be used. With this approach only the output of the segmentation process should respect the defined format, and users are free to adopt their own algorithms. The `DOCRegionDescription` is a modified version of the `RichRegionDescription` [1], including specific attributes for a document recognition environments.

The XML Schema of the `DOCRegionDescription` is reported in the Appendix. A set of standard attributes, namely `NumPixels`, `Barycenter`, `Size`, `MeanColor`, `Texture`, `RegionContour`, `Importance` and `Shape`, are used in order to satisfy general features of segmentation algorithms. Moreover two specific attributes are important: the `Interpreted` attribute, and the `Type` (which can be `Auto` or `Manual`) of the `Segmentation` tag.

The `Interpreted` attribute is valorized from the segmentation algorithm if it performs also classification (text, image or graph region). If the algorithm provides a method for evaluating its result, e.g., by using a range of values for the acceptance test of the segmented regions, the `Type` attribute helps: if the values are acceptable the `Type` value should be `Auto` and the next phase will be indexing, while for that documents where the values are out of range should be `Manual` and the next phase will be the annotation by the user. Of course, if we want to perform only manual annotation, the `Type` attribute should be fixed to `Manual`.

3 Details of the Different Components

In this section we provide some details on the main JDAN components.

3.1 The Recognition Server

The Recognition Server performs the automatic document recognition and indexing. Users can choose their own segmentation method or algorithm; what is required, in order to be compliant with the framework, is to produce an XML

[4] In our case we have used our algorithms for writing the segmentation and classification engine, but any other segmentation and classification application compliant with the `DOCRegionDescription` format could be written on top of the framework.

output compliant with the `DOCRegionDescription`. In the prototype we are currently implementing, the Recognition Server includes four main subcomponents [1–3]: *(i)* the preprocessor, *(ii)* the split module, *(iii)* the merge module and *(iv)* the classification module.

The preprocessor is based on the computation of the color histogram of the region and is used to let the other phases get the values computed by this module, in order to make the right decisions [4]. Moreover in this phase the original image of the document is loaded into main memory in order to obtain better computation performances.

The split module takes input from the preprocessing phase and applies a particular quad-tree technique in order to split the document into small blocks. Then the split module passes its result to the merge module, which applies preclassification criterion in order to merge the similar regions into big regions [5, 6]. We use local operators with variable threshold in order to compute this phase.

Finally using global operators we have the real engine of this system in the classification module that computes the classification procedure according to the classification logic. It outputs the classified document, with its different regions, highlighted and interpreted, in XML format. If there is at least one region "not interpreted", the `Type` in the XML document will be switched to `Manual` and the Data-Entry Plug-in is invoked, only for those "not interpreted" regions.

Following we show an example of an XML region description automatically extracted from the document shown in Figure 2.

```
<?xml version="1.0" encoding="UTF-8"?>
<DocRegionDescription xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="...\DELOSWorkshop2004\DocRegionDescription_++_XSD.xsd"
FileName="ta00004bw.gif" Height="528" Width="383">
   <Segmentation Param1="0.05" Param2="3000" Param3="200" Type="Manual" Algo="anAlgorithm">
      <Region ID="0" Type="TEXT" KeyWords="Note">
         <Size x1="215" y1="503" x2="343" y2="511"/>
      </Region>
      <Region ID="1" Type="TEXT" KeyWords="Introduction">
         <Size x1="35" y1="243" x2="343" y2="494"/>
      </Region>
      <Region ID="2" Type="GRAPH">
         <Size x1="38" y1="57" x2="328" y2="177"/>
         <ConnectivityNumber EuleroNumber="371"/>
         <Barycenter x="38" y="57"/>
         <Elongation Elongation="2"/>
      </Region>
      <Region ID="3" Type="TEXT" KeyWords="#">
         <Size x1="37" y1="12" x2="343" y2="36"/>
      </Region>
      <Region ID="4" Type="TEXT" KeyWords="#">
         <Size x1="35" y1="197" x2="195" y2="230"/>
      </Region>
   </Segmentation>
</DocRegionDescription>
```

3.2 The Data-Entry Plug-in

The recognition phase is based on the segmentation and classification module, which takes as input a document page, which is then presented to the user if manual indexing was decided (by the user, or by the "not interpreted" parameter) through the Data-Entry Plug-in (a Java applet).

The Data-Entry Plug-in allows users to evaluate the automatic classification performed by the system and edit the segmentation for indexing; users can also edit the recognized regions by the classification engine and adjust their values and sizes. The output of this phase is an XML file that will be imported in the Indexing Module for indexing and querying. Figure 2 shows the Data-Entry module.

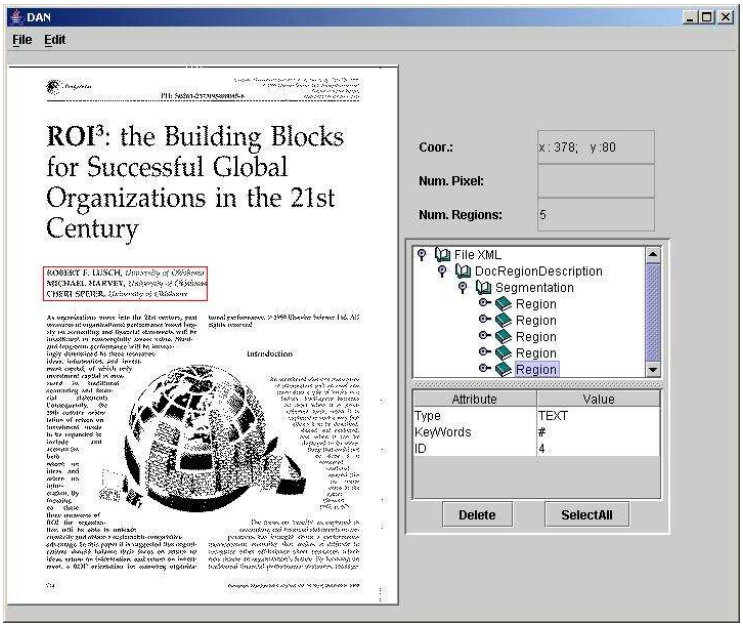


Fig. 2. The Data-Entry Plug-in

3.3 The Indexing Server

The indexing server is based on a multi-tier architecture. It is an EJB-based system, namely JBoss 3.0. The database layer stores and accesses all of the basic information kept in JDAN, such as text labels or segmentation and indexing features. The web layer is responsible for turning the data fragments stored in the database into useful forms that are presented to the web client, e.g., a report or a query result. The web client presents forms to users, and submits data back to the web layer.

3.4 Archive Release

The final stage in the recognition process could be to release each document to a content management or workflow system. In the release process, the image

files are written to permanent storage and the data is written to the target database or content management software. When dealing with forms, extracted form data can be released to a back-end database or line of business application. In addition, the Archive Release module allows users to write their own custom release modules, either to modify the standard release procedure or to release documents into a proprietary back-end or non relational database.

4 Deploying the JDAN Framework and Final Remarks

In this section we show a typical use case of an application built on top of the JDAN framework, and then we discuss about possible configurations supporting the presented framework.

In Figure 3, a typical use case of an application adopting JDAN is presented. After a manual or automatic indexing of a batch of documents, a set of indexes compliant with the `DOCRegionDescription` format are sent to the Indexing Server (step *(1)*); by using the indexing server such indexes are interpreted and then stored in the database (step *(2)*). When a request from the web client is received, an SQL query is performed on the database and the indexing server formats the result in order to transform it in HTML/XML (step *(3)*); the HTML/XML query results are then sent via HTTP to the web client for user presentation (step *(4)*).

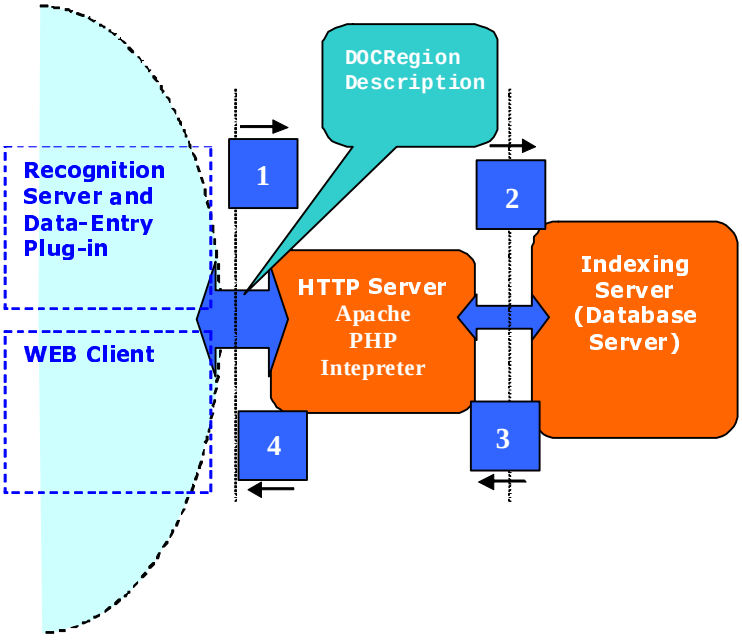


Fig. 3. A typical use of the JDAN framework

A typical configuration would have the Recognition Server running onto a server cluster, with many Data-Entry Plug-ins running onto different clients. This configuration defines a document capture center, that could be offered in a complex grid as a basic service; in another center, an Indexing cluster is offered as another service, and different Web clients could perform queries onto the already captured and indexed documents requesting information to the Indexing center.

In Figure 4 we show an example of a query made with an Applet module. The query is based on previous XML description of the document automatically extracted by the recognition server.

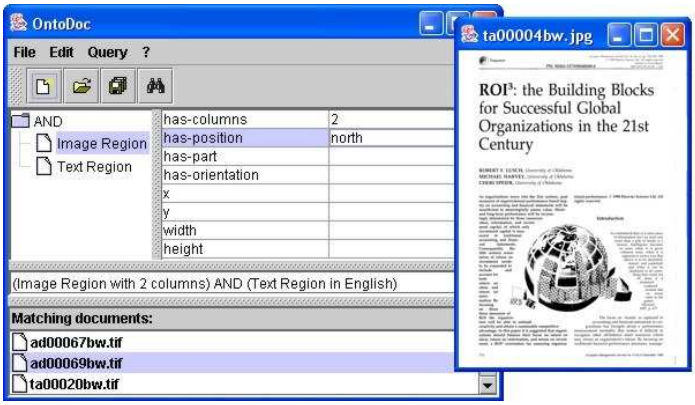


Fig. 4. Querying JDAN for retrieving a document by using previously extracted XML document region description

As final remarks, we would like to point out that the proposed architecture and framework aim at defining an open environment for assisting users in recognizing multimedia documents also using segmentation algorithms for classifying image regions, and image retrieval algorithms. Whereas current available environments are proprietary and closed, the definition of an XML-based interchange format allows to suitable assemble different component-based technologies in order to define a complex framework.

JDAN should be considered as a preliminary step in the direction of multimedia document managing standard framework with region segmentation and classification, thus aiming at automatic recognition of image database and batch acquisition of multiple multimedia documents types and formats. Finally JDAN could be considered as a test bed for students and researchers that are projecting and implementing algorithms for document analysis and document capture.

Acknowledgements

This work was carried out in the context of the FP6 Network of Excellence on Digital Libraries (DELOS, `http://www.delos.info/`).

References

1. L. Cinque, S. Levialdi, A. Malizia, F. De Rosa. DAN: an Automatic Segmentation and Classification Engine for Paper Documents. In *Proc. 5th International Workshop on Document Analysis Systems (DAS 2002)* (Princeton, NY, USA, 2002), Springer Verlag LNCS 2423.
2. L. Cinque, S. Levialdi, A. Malizia, F. De Rosa. A Visual Query-by-example System for Digital Documents. In *Proc. IEEE Symposium on Human Centric Computing Languages and Environments (HCC'03)* (Auckland, New Zealand, 2003), IEEE CS Press.
3. L. Cinque, S. Levialdi, A. Malizia. An Integrated System for the Automatic Segmentation and Classification of Documents. In *Proc. IASTED International Conference on Signal Processing, Pattern Recognition and Applications (SPPRA 2002)* (Creete, Greece, 2002),
4. G. Nagy, S. Seth, M. Viswanathan. A Prototype Document Image Analysis System for Technical Journals. *Computer*, 25(7): 10 – 22, July 1992.
5. T. Ojala, M. Pietikainen. Unsupervised Texture Segmentation Using Feature Distributions. *Pattern Recognition*, vol. 32: 477 – 486, 1999.
6. M. Span, R. Wilson. A Quad-Tree Approach to Image Segmentation Which Combines Statistical and Spatial Information. *Pattern Recognition*, vol. 18: 257 – 269, 1985.

Appendix. The `DOCRegionDescription` XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
   <xs:complexType name="BarycenterType">
      <xs:attribute name="x" type="xs:integer" use="required"/>
      <xs:attribute name="y" type="xs:integer" use="required"/>
   </xs:complexType>
   <xs:complexType name="ConnectivityNumberType">
      <xs:attribute name="EuleroNumber" type="xs:integer" use="required"/>
   </xs:complexType>
   <xs:element name="DocRegionDescription">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="Segmentation" type="SegmentationType" maxOccurs="unbounded"/>
         </xs:sequence>
         <xs:attribute name="Version" type="xs:string"/>
         <xs:attribute name="FileName" type="xs:string" use="required"/>
         <xs:attribute name="Height" type="xs:integer" use="required"/>
         <xs:attribute name="Width" type="xs:integer" use="required"/>
      </xs:complexType>
   </xs:element>
   <xs:complexType name="ElongationType">
      <xs:attribute name="Elongation" type="xs:integer" use="required"/>
   </xs:complexType>
   <xs:complexType name="ImportanceType">
      <xs:attribute name="value" type="xs:string" use="required"/>
   </xs:complexType>
   <xs:complexType name="MeanColorType">
      <xs:attribute name="red" type="xs:float" use="required"/>
      <xs:attribute name="green" type="xs:float" use="required"/>
      <xs:attribute name="blue" type="xs:float" use="required"/>
   </xs:complexType>
   <xs:complexType name="RegionType">
      <xs:choice maxOccurs="unbounded">
         <xs:element name="ConnectivityNumber" type="ConnectivityNumberType"/>
```

```
            <xs:element name="Barycenter" type="BarycenterType"/>
            <xs:element name="Size" type="SizeType"/>
            <xs:element name="MeanColor" type="MeanColorType"/>
            <xs:element name="Texture" type="TextureType"/>
            <xs:element name="RegionContour" type="RegionContourType"/>
            <xs:element name="Importance" type="ImportanceType"/>
            <xs:element name="Shape" type="ShapeType"/>
            <xs:element name="Elongation" type="ElongationType"/>
         </xs:choice>
         <xs:attribute name="ID" type="xs:string" use="required"/>
         <xs:attribute name="Type" use="required">
            <xs:simpleType>
               <xs:restriction base="xs:NMTOKEN">
                  <xs:enumeration value="TEXT"/>
                  <xs:enumeration value="GRAPH"/>
                  <xs:enumeration value="IMAGE"/>
                  <xs:enumeration value="BACKGROUND"/>
                  <xs:enumeration value="TBD"/>
                  <xs:enumeration value="ERROR"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:attribute>
         <xs:attribute name="KeyWords" type="xs:string"/>
      </xs:complexType>
      <xs:complexType name="RegionContourType">
         <xs:attribute name="Length" type="xs:integer" use="required"/>
         <xs:attribute name="PixelList" type="xs:string" use="required"/>
      </xs:complexType>
      <xs:complexType name="SegmentationType">
         <xs:sequence>
            <xs:element name="Region" type="RegionType" maxOccurs="unbounded"/>
         </xs:sequence>
         <xs:attribute name="Type">
            <xs:simpleType>
               <xs:restriction base="xs:NMTOKEN">
                  <xs:enumeration value="Auto"/>
                  <xs:enumeration value="Manual"/>
               </xs:restriction>
            </xs:simpleType>
         </xs:attribute>
         <xs:attribute name="Param1" type="xs:string"/>
         <xs:attribute name="Param2" type="xs:string"/>
         <xs:attribute name="Param3" type="xs:string"/>
         <xs:attribute name="Algo" type="xs:string"/>
      </xs:complexType>
      <xs:complexType name="ShapeType">
         <xs:attribute name="Shape" type="xs:string" use="required"/>
      </xs:complexType>
      <xs:complexType name="SizeType">
         <xs:attribute name="x1" type="xs:integer" use="required"/>
         <xs:attribute name="y1" type="xs:integer" use="required"/>
         <xs:attribute name="x2" type="xs:integer" use="required"/>
         <xs:attribute name="y2" type="xs:integer" use="required"/>
      </xs:complexType>
      <xs:complexType name="TextureType">
         <xs:attribute name="p1" type="xs:string" use="required"/>
         <xs:attribute name="p2" type="xs:string" use="required"/>
      </xs:complexType>
   </xs:schema>
```

Analysis and Evaluation of Service Oriented Architectures for Digital Libraries

Hussein Suleman

Department of Computer Science, University of Cape Town
Private Bag, Rondebosch, 7701, South Africa
hussein@cs.uct.ac.za

Abstract. The Service Oriented Architecture (SOA) that underlies the Web Services paradigm of computing is widely regarded as the future of distributed computing. The applicability of such an architecture for digital library systems is still uncertain, as evidenced by the fact that virtually none of the large open source projects (e.g., Greenstone, EPrints, DSpace) have adopted it for internal component structuring. In contrast, the Open Archives Initiative (OAI) has received much support in the DL community for its Protocol for Metadata Harvesting, one that in principle falls within the scope of SOA. As a natural extension, the Open Digital Library project carried the principles of the OAI forward into a set of experimental derived and related protocols to create a testbed for component-based digital library experiments. This paper discusses a series of experiments with these components to confirm that SOA and a service-oriented component architecture is indeed applicable to digital library systems, by evaluating issues of simplicity and understandability, reusability, extensibility and performance.

1 Introduction

Service-oriented computing is a relatively new paradigm of computing where tasks are subdivided and performed by independent and possibly remote components that interact using well-defined communications protocols [19]. In particular, the Service-Oriented Architecture (SOA) refers to a framework built around XML and XML messaging, with standards for how messages are encoded, how protocol syntax is specified and where instantiations of services are to be located. These are exemplified by the SOAP [4], WSDL [2] and UDDI [1] specifications respectively. It is often argued that SOA can be adopted by an organisation to increase reuse, modularity and extensibility of code, while promoting a greater level of interoperability among unrelated network entities.

From a somewhat different perspective, the Open Archives Initiative (OAI) attempted to address interoperability first and foremost, by designing a protocol for the efficient incremental transfer of metadata from one network entity to another. This Protocol for Metadata Harvesting (PMH) [7] has since been adopted by many large digital archives and has become the primary mechanism for digital library interoperability in 2004. The OAI-PMH is very closely related

to SOA as it adopts a Web-distributed view of individual systems, where independent components - listed on the OAI website - interact through the medium of a well-specified protocol and XML-encoded messages. The OAI-PMH differs significantly from the SOA in that it is more concerned with a specific set of protocol/encoding semantics while SOA specifies only an underlying transport mechanism that could be applied to many different protocol suites. In this sense, a marriage of OAI-PMH and SOA is both possible and probable [3].

In the interim, however, one of the reasons OAI-PMH has not as yet migrated to SOA is that the technology is not sufficiently well proven. In addition, OAI-PMH is aimed at interaction among entire systems, viewed as components of a super-system. SOA, however, is easily applied to a finer granularity, where reuse and modularity of components are crucial. To bridge this gap, and translate the core principles of OAI-PMH to fine-grained interaction among small components of a larger digital library, the Open Digital Library (ODL) project defined a suite of protocols based on the widely accepted principles of OAI-PMH, but aimed at the needs of digital library subsystem interaction [13][14].

To maintain thematic consistency, these protocols were designed in an object-oriented fashion, where each protocol built on a previous one, extending and overriding semantics as needed. A set of reference implementations of components were then created to provide the following services: **searching**; **browsing**; tracking of **new items**; **recommendation** by collaborative filtering; **annotation** of items as an independent service; numerical **ratings** for items in a collection; **merging** of sub-collections; and **peer review** workflow support.

Finally, these components were analysed and evaluated to determine how applicable a fine-grained component model is to the construction of digital libraries, and possibly expose problems and shortcomings to be addressed in future research. At the same time, the results obtained are generalisable to Web-based systems and are indicators for the success of SOA as it is applied to digital library componentisation.

2 Experiments: Simplicity and Understandability

The first set of experiments aimed to determine if components with Web-based interfaces can be composed into complete systems with relative ease by non-specialists. Three different user communities were introduced to the underlying principles of OAI and the component architecture devised and were then led through the procedure of building a simple digital library system using the components.

2.1 OSS4LIB

The first group, at an ALA OSS4LIB workshop, provided anecdotal evidence that the component-connection approach to building DLs was feasible. The approximately 12 participants were largely technical staff associated with libraries and therefore had minimal experience with installation of software applications.

They were carefully led through the process of configuring and installing multiple components and were pleasantly surprised at the ease with which custom digital libraries can be created from components. This led to a second, more controlled, experiment as detailed below.

2.2 Installation Test

The second group comprised 56 students studying digital libraries. The aim of this study was to gauge their level of understanding of OAI and ODL components and their ability to complete a component composition exercise satisfactorily. The objective was to install the following components and link them together to form a simple digital library, as illustrated also in Fig 1:

- XMLFile: a simple file-based OAI archive
- Harvester: an OAI/ODL harvester
- IRDB: an ODL search engine
- IRDB-ui: a simple user interface for IRDB

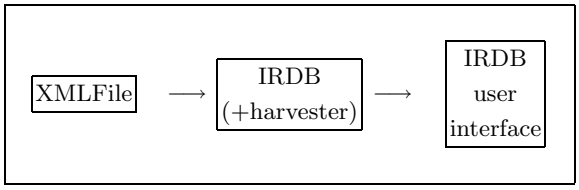


Fig. 1. Architecture of simple componentised digital library

Students were given an hour-long introduction to OAI and ODL and then given detailed instructions to perform the component composition exercise. Upon completion, they were asked to fill out a questionnaire to evaluate the experience of building this system from components. Table 1 displays a summary of the responses to the core questions asked of users.

In addition to these questions, typical demographic information was collected to ascertain skills levels and exposure to the technical elements of the experiment. The different backgrounds of participants evident from the responses to demographic questions makes it difficult to analyse these results without taking into account all of the interaction effects that result from past experience. Since OAI and ODL utilise various different Web technologies, it is non-trivial to enumerate all of the pre-requisites and determine their independent effects. It may be possible to construct an experimental model to minimise the interaction effects, but this will require finding unique participants, each with a very particular background and training. This may prove difficult because of the cutting-edge nature of Web-based technology. Taking these difficulties into account, any analysis of such an experiment cannot easily determine general trends.

Table 1. Summary of Responses to Component Composition Experiment

Question/Response	S. Agree	Agree	Neutral	Disagree	S. Disagree
Understand concepts of OAI-PMH.	9	38	9		
Understand concepts of ODL.	6	36	14		
Instructions were understandable.	35	20	1		
Installing components was simple.	36	18	2		
Configuration was simple.	33	21	2		
Connecting Harvester+XMLFile was simple.	28	25	3		
Connecting IRDB+XMLFile was simple.	26	25	5		
Understanding of OAI/ODL has improved.	13	25	12	6	
I will use ODL and OAI components.	12	33	10	1	

Nine respondents who indicated that they did not know how Web Services worked answered affirmative when asked if they had done Web Services-related development. This may be because they interpreted the question as referring to Web-related services other than SOAP, WSDL, and UDDI, or because they had done development work without understanding the underlying standards and information model of Web Services. Either of these is consistent with the vague understanding many people have of Web Services.

Judging from the responses to the first two questions in Table 1, most participants appear to have grasped the basic concepts related to OAI and ODL. The fact that some participants were unsure indicates that an hour and 15 minutes may not be enough for a person building a digital library to learn enough about OAI and ODL. This raises the question of just how much training a person needs before being able to effectively use OAI and ODL technology (or any Web-service-related technology). Also, more of the participants were able to understand OAI rather than ODL; this is expected since ODL builds on OAI.

Most participants agreed (or strongly agreed) that the instructions were understandable. The instructions were very detailed so that even if participants did not understand one section of the exercise, they were still able to complete the rest of the steps.

Installation and configuration of individual components as well as interconnecting different components was deemed to be simple. As these are two basic concepts underlying ODL (that all services can be independent components and that systems are built by interconnecting service components), it supports the hypothesis of this experiment that ODL is simple to understand and adopt - which commutes to Web Services.

There was not much agreement about the ability of the exercise to improve the participants understanding of OAI and ODL. This can be attributed to the sheer volume of new concepts covered during the presentation and exercise. Given that approximately half of the participants had never created a CGI-based Web application before, the learning curve was quite steep. In practice, those who adopt OAI and ODL technology are usually digital library practitioners who already have experience with the construction of dynamic Web-based information systems.

In spite of all these factors, two-thirds of the participants indicated an interest in using similar components if they have a need for such services. Thus, even without a thorough understanding of the technology and other available options, the simple and reusable nature of the components seemed to appeal to participants.

Thirteen participants provided optional feedback in the survey, and these ranged from positive to somewhat skeptical. Eight of the reactions were positive, including the following comments:

- "I think the idea is very good and the approaches to build digital libraries is easy."
- "They provide a way to get up and running very quickly with a Web application."

Some participants were not sure about the workflow as indicated by the comment:

- "We have high level idea but detailed explanation will be great."

One comment in reference to the questions on simplicity of installation and configuration included:

- "I dont know; custom config might not be simple."

This summarises the notion that components should be simple enough to bootstrap a development process but still powerful enough to support a wide range of functionality. In particular, the above comment refers to the XMLFile component that is simple to install and use in its default configuration, but can be non-trivial to configure if the records are not already OAI-compatible. In such a situation, XSL transformations can be used to translate the records into acceptable formats. However, irrespective of the complexity of configuration for a particular instance, the OAI/ODL interface to such components always is the same.

Some questions raised during the lab sessions revealed very important issues that need to be addressed in future development of ODL or related Web-based standards:

- Confusion over baseURLs
  - Some participants were confused regarding which baseURL to use in which instance. Since all URLs were similar, it was not obvious – this ought not to happen in practice with any system built on OAI, ODL or Web Services technology.

  - Entering URLs by hand resulted in many typographical errors. Ideally, such links must be made using a high-level user interface that masks complex details like URLs from the developers.
  - The user interface was sometimes connected to the wrong component. While it is possible for a user interface to Identify the service component before using it, this will be inefficient. As an alternative, user interfaces can themselves be components, with associated sanity tests applied during configuration.
- Failures during harvesting
  - Harvesting will fail if the baseURL is incorrect, but there are no obvious graceful recovery techniques. The components used in the experiment assume a catastrophic error and stop harvesting from the questionable archive pending user intervention. Better algorithms can be devised to implement exponential back-off and/or to trigger notification of the appropriate systems administrator.

2.3 Comparison Test

Finally, a third experiment was conducted to contrast the component approach to system building with the traditional monolithic system approach. 28 students in digital libraries were asked to install a system similar to the one in the previous experiment as well as a version of the Greenstone [18] system, and compare and contrast them from the perspectives of ease of use and installation.

The responses highlighted both positive and negative aspects of both systems. The majority of respondents indicated that Greenstone was easier to install, being a single cohesive package. However, it was also agreed by almost all respondents that the component approach was more flexible and powerful, and therefore applicable to a larger set of problem domains than the monolithic equivalent. There was tension between the higher degree of architectural control possible with ODL and the increase in complexity it introduced for those not wanting such control. The service-oriented approach was also preferred for its scalability, genericity and support for standards, which was not as evident in the monolithic approach. A number of respondents were undecided as to an outright preference, given that each solution had its advantages and disadvantages - leading to the conclusion that an ideal solution would capitulate on the strengths of both approaches, somehow giving end users the advantages of component-based customisation and flexibility as well as the advantages of cohesion and simplicity inherent in the non-component approach.

3 Experiments: Reusability and Extensibility

To test for reusability and extensibility, the suite of components was made available to colleagues for integration into new and existing systems. A number of digital library systems have since made use of the components, either directly, or composed/aggregated into other components. The following are a discussion

of how some projects have integrated service-oriented components and protocols into their architectures.

3.1 AmericanSouth.org

AmericanSouth.org [6] is a collaborative project led by Emory University to build a central portal for scholarly resources related to the history and culture of the American South. The project was initiated as a proof-of-concept test of the metadata harvesting methodology promoted by the OAI. Thus, in order to obtain data from remote data sources, the project relies mainly on the OAI-PMH.

The requirements for a central user portal include common services such as searching and browsing. AmericanSouth.org used ODL components to assist in building a prototype of such a system. The DBUnion, IRDB and DBBrowse components were used in addition to XMLFile and other custom-written OAI data provider interfaces. Many questions about protocol syntax and component logic were raised and answered during the prototyping phase, suggesting that more documentation is needed. Alternatively, pre-configured networks of components can be assembled to avoid configuration of individual components. Both of these approaches are being investigated in the DL-in-a-Box project [10].

The production system for AmericanSouth.org still uses multiple instantiations of XMLFile but the ODL components have been replaced with the ARC search engine [9] largely because of concerns over execution speed of the IRDB search engine component. This in itself indicates the ease with which service-oriented components can be replaced in a system whose requirements change over time.

3.2 CITIDEL

CITIDEL - the Computing and Information Technology Interactive Digital Education Library [5] - is the computing segment of NSF's NSDL - the National Science, Technology, Engineering and Mathematics Digital Library [8]. CITIDEL is building a user portal to provide access to computing-related resources garnered from various sources using metadata harvesting wherever possible. This user portal is intended to support typical resource discovery services, such as searching and category-based browsing, as well as tools specific to composing educational resources, such as lesson plan editors.

From the initial stages, CITIDEL was envisioned as a componentised system, with an architecture that evolves as the requirements are refined. The initial system was designed to include multiple sources of disparate metadata and multiple services that operate over this data, where each data source and service is independent.

CITIDEL uses components from various sources. In terms of ODL, this includes the IRDB and Thread components to implement simple searching and threaded annotations, respectively. The IRDB component was modified to make more efficient use of the underlying database, but the interface was unchanged.

3.3 BICTEL/e

The BICTEL/e project, led by the Universite Catholique de Louvain, is building a distributed digital library of dissertations and e-prints within the nine French-speaking universities in Belgium. The project adopted use of OAI and ODL components to support dissertations and e-prints collections at each university and at a central site, alongside some non-ODL components.

3.4 Sub-classing

Some component implementations were created by sub-classing existing components. All of the component modules were written in object-oriented Perl, which allows for single inheritance, so this was exploited when possible. Since the DBRate and DBReview components also store the original transaction records submitted to them, they were derived from the Box component. In each case, some of the methods were overridden to provide the necessary additional functionality.

3.5 Layering: VIDI

The VIDI project [17] developed a standard interface, as an extension of the OAI protocol, to connect visualisation systems to digital libraries. A prototype of the VIDI reference implementation links into the search engine of the ETD Union Catalog [15] to obtain search results. The search engine used in the ETD Union Catalog understands the ODL-Search protocol. Thus, additional services are provided as a layer over an ODL component, without any reciprocal awareness necessary in the ODL system.

3.6 Layering: MAIDL

MAIDL, Mobile Agents In Digital Libraries [12], is a federated search system connecting together heterogeneous Web-accessible digital libraries. The project uses the "odlsearch1" syntax, as specified in the ODL-Search protocol, in order to submit queries to its search system. Further communication among the mobile agents and data providers transparently utilize the XOAI-PMH protocol [14].

4 Experiments: Performance

Lastly, performance tests were conducted to determine the effect of Web-based inter-component communication. Measurements were taken for heavily loaded systems, systems that rely on multiple components to respond to requests (e.g., portals) as well as the contribution made to system latency by different layers in the architecture.

The most critical of measurements looked at the effect of additional Web-application layering on the execution times of individual components of a larger

system. The IRDB search engine component was used for this test because search operations take a non-trivial (and therefore measurable) amount of time and the pre-packaged component includes a direct interface to the search engine that allows bypassing of the ODL protocol layer.

For test data, a mirror of the ETD Union Archive was created and this then was harvested and indexed by an instance of the IRDB component. 7163 items were contained in this collection, each with metadata in the Dublin Core format.

The test was to execute a search for a given query. Three queries were used: "computer science testing", "machine learning", and "experiments". At most the first 1000 results were requested in each case. Each query was executed 100 times by a script to minimise the effect of the script on the overall performance. The first run of each experiment was discarded to minimise disk access penalties, and an average of the next 5 runs was taken in each case.

Six runs were made for each query:

1. Executing lynx to submit a ListIdentifiers query through the Web server interface.
2. Executing wget to submit a ListIdentifiers query through the Web server interface.
3. Using custom-written HTTP socket code to submit a ListIdentifiers query through the Web server interface.
4. Executing the search script directly from the command-line, thereby bypassing the Web server.
5. Executing testsearch.pl to bypass both the Web server and the ODL layer.
6. Using direct API calls to the IR engine, without spawning a copy of testsearch.pl in each iteration.

The time was measured as the "wall-clock time" reported by the bash utility program "time" from the time a run started to the time it ended. The script that ran the experiment controlled the number of iterations (100, in this case) and executed the appropriate code in each of the 6 cases above. In each case, the output was completely collected and then immediately discarded - thus, each iteration contributed the complete time between submitting a request and obtaining the last byte of the associated response, hereafter referred to as the execution time.

It was noticable from the measured times that execution time increases as more layers are introduced into the component. This increase is not always a large proportion of the total time, but the difference between Test-1 and Test-6 is significant. The time differences between pairs of consecutive tests is indicated in Table 2.

Test-1, Test-2 and Test-3 illustrate the differences in times due to the use of different HTTP clients. In Test-1, the fully-featured text-mode Web browser lynx was used. In Test-2, wget was used instead, and the performance improved because wget is a smaller application that just downloads files. Test-3 avoided the overhead of spawning an external client application altogether by using custom-written network socket routines to connect to the server and retrieve responses

Table 2. Time differences between pairs of consecutive tests

Query	Test1-2	Test2-3	Test3-4	Test4-5	Test5-6
"computer science testing"	4.52	1.04	0.67	0.33	8.57
"machine learning"	3.72	0.63	0.58	0.22	10.62
"experiments"	4.26	0.94	0.35	0.66	8.53

to requests. The differences are only slight but there is a consistent decrease for all queries.

The difference between Test-3 and Test-4 is due to the effect of requests and responses passing through the HTTP client and the Web server. While no processes were spawned at the client side in Test-3, a process was still spawned by the Web server to handle each request at the back-end. This script was run directly in Test-4, so the difference in time is due solely to the request being routed through the Web server. This difference is small, so it suggests that the Web server does not itself contribute much to the total execution time.

The difference between Test-4 and Test-5 is due to the ODL-Search software layer that handles the marshalling and unmarshalling of CGI parameters and the generation of XML responses from the raw list of identifiers returned by the IR engine. This difference is also small, indicating that the additional work done by the ODL layer does not contribute much to the total time of execution.

The difference between Test-5 and Test-6 is due to the spawning of a new process each time the IRDB component is used. This difference is substantial and indicates that process startup is a major component of the total execution time.

In general, the execution times for the IRDB component (as representative of ODL components in general) were much higher than the execution times for direct API calls. However, this difference in execution time is due largely to the spawning of new processes for each request. The ODL layer and the Web server contribute only a small amount to the total increase in execution time. As a follow-on experiment, different Web application acceleration technologies were used to verify that no substantial execution speed penalty need be incurred when adopting a service-oriented component approach to system development (details of this experiment can be found in [16]). This illustrated, in particular, that there is not a significant difference between using direct API calls and invoking a Web service. Ultimately, these experiments confirm that there is little cause for concern, performance-wise, if Web technology is chosen wisely – for example, using persistent Web applications such as servlets for Java applications or SpeedyCGI for Perl applications.

5 Conclusions

The Service Oriented Architecture is still a fairly new concept in DL systems, with most systems supporting one or two external interfaces, for example OAI-PMH. This work has investigated the applicability of SOA as a fundamental architecture within the system, an analysis of which has demonstrated its feasibility according to multiple criteria, while exposing issues that need to be considered in future designs.

6 Future Work

The most important aspect highlighted by ongoing experiments was the need for better and simpler management of components, so that the complexity of deconstructing a monolithic system into service-oriented components did not fundamentally increase the complexity of overall system management. To this end, the ongoing "Flexible Digital Libraries" project is investigating how external interfaces can be defined for remote management of components, thus enabling automatic aggregration and configuration of components by installation managers and real-time component management systems. The first of such systems to be built, BLOX [11], allows a user to build a system visually using instances of Web-accessible components residing on remote machines. Ongoing work is looking into how systems built with such an interface can be packaged and redeployed, thus bridging the gap between components and monolithic systems from a system installation perspective.

This work naturally lends itself to a future architecture that allows migration and replication of components to support "component farms" as a replacement of "server farms", as well as peer-to-peer and grid computing paradigms, where services are needs-based and location-independent.

At the same time, some effort needs to go into how services are orchestrated and composed/aggregrated at a higher level to perform useful functions needed by users. The WS-Flow and WS-Choreography activities are useful starting points but more investigation is needed into their suitability for integration with user interface and workflow design as a front-end to the component farms envisioned as the back-end of future Web-based information applications.

References

1. Ariba, Inc., IBM and Microsoft (2000), UDDI Technical White Paper, 6 September 2000. Available http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf
2. Christensen, E., F. Curbera, G. Meredith and S. Weerawarana (2001), Web Services Description Language (WSDL) 1.1, W3C. Available http://www.w3.org/TR/wsdl
3. Congia, Sergio, Michael Gaylord, Bhavik Merchant and Hussein Suleman (2004), "Applying SOAP to OAI-PMH", to appear in Proceedings of ECDL2004, Bath, UK, 12-17 September 2004.

4. Gudgin, M., M. Hadley, N. Mendelsohn, J. Moreau and H. F. Nielson (2003), SOAP Version 1.2 Part 1: Messaging Framework and Part 2: Adjuncts, W3C, 24 June 2003. Available http://www.w3.org/TR/2003/REC-soap12-part1-2003-0624/ and http://www.w3.org/TR/2003/REC-soap12-part2-2003-0624/
5. Fox, Edward A., Deborah Knox, Lillian Cassel, John A. N. Lee, Manuel Pérez-Quiñones, John Impagliazzo and C. Lee Giles (2002), CITIDEL: Computing and Information Technology Interactive Digital Educational Library. Website http://www.citidel.org
6. Halbert, M. (2002), AmericanSouth.org. Website http://www.americansouth.org
7. Lagoze, Carl, Herbert Van de Sompel, Michael Nelson and Simeon Warner (2002), The Open Archives Initiative Protocol for Metadata Harvesting Version 2.0, Open Archives Initiative, June 2002. Available http://www.openarchives.org/OAI/2.0/openarchivesprotocol.htm
8. Lagoze, Carl, Walter Hoehn, David Millman, William Arms, Stoney Gan, Dianne Hillmann, Christopher Ingram, Dean Krafft, Richard Marisa, Jon Phipps, John Saylor, Carol Terrizzi, James Allan, Sergio Guzman-Lara and Tom Kalt (2002), "Core Services in the Architecture of the National Science Digital Library (NSDL)", in Proceedings of Second ACM/IEEE-CS Joint Conference on Digital Libraries, Portland, OR, USA, 14-18 July 2002, pp. 201-209.
9. Liu, Xiaoming, Kurt Maly, Mohammad Zubair and Michael L. Nelson (2001), "Arc: an OAI service provider for cross-archive searching", in Proceedings of First ACM/IEEE-CS Joint Conference on Digital Libraries, Roanoke, VA, USA, 24-28 June 2001, pp. 65-66.
10. Luo, Ming (2002), Digital Libraries in a Box. Website http://dlbox.nudl.org
11. Moore, David, Stephen Emslie and Hussein Suleman (2003), BLOX: Visual Digital Library Building, Technical Report CS03-20-00, Department of Computer Science, University of Cape Town. Available http://pubs.cs.uct.ac.za/
12. Nava Muñoz, and Sandra Edith (2002), Federación de Bibliotecas Digitales utilizando Agentes Móviles (Digital Libraries Federation using Mobile Agents), Master's thesis, Universidad de las Américas Puebla.
13. Suleman, Hussein, and Edward A. Fox (2001), "A Framework for Building Open Digital Libraries", in D-Lib Magazine, Vol. 7, No. 12, December 2001. Available http://www.dlib.org/dlib/december01/suleman/12suleman.html
14. Suleman, H., and E. A. Fox (2002), "Designing Protocols in Support of Digital Library Componentization", 6th European Conference on Research and Advanced Technology for Digital Libraries (ECDL2002), Rome, Italy, 16-18 September 2002.
15. Suleman, H., and E. A. Fox (2002), "Towards Universal Accessibility of ETDs: Building the NDLTD Union Archive", Fifth International Symposium on Electronic Theses and Dissertations (ETD2002), Provo, Utah, USA, 30 May-1 June 2002.Available http://www.husseinsspace.com/publications/etd_2002_paper_union.pdf
16. Suleman, H. (2002), Open Digital Libraries, Ph.D. dissertation, Virginia Tech. Available http://scholar.lib.vt.edu/theses/available/etd-11222002-155624/
17. Wang, J. (2002), A Lightweight Protocol Between Visualization Tools and Digital Libraries, Master's Thesis, Virginia Polytechnic Institute and State University.
18. Witten, I. H., R. J. McNab, S. J. Boddie and D. Bainbridge (2000), "Greenstone: A Comprehensive Open-Source Digital Library Software System", in Proceedings of Fifth ACM Conference of Digital Libraries, San Antonio, Texas, USA, 2-7 June 2000, pp. 113-121.
19. Yang, J. (2003), "Web Service Componentization", in Communications of the ACM, Vol. 46, No. 10, October 2003, ACM Press, pp. 35-40.

Towards a Global Infrastructure for Georeferenced Information

Michael Freeston

Research Coordinator, Alexandria Digital Library Project
University of California Santa Barbara USA
freeston@alexandria.ucsb.edu

Abstract. This paper aims to encourage an appreciation of the value of georeference in digital libraries as a complementary search paradigm to more conventional text-based information retrieval and indexing techniques. In particular, it discusses the concept of the Digital Earth, and analyses why this concept has so far failed to make the transition from a research idea to a viable knowledge discovery tool. The analysis leads to the identification of a wider and more fundamental issue: the place of digital libraries in the technical and administrative infrastructure of the next generation of the World Wide Web – the so-called Cyberinfrastructure. The argument is made that it is the Grid community which is currently taking the lead in defining this infrastructure, and that future digital library research – including the development of a global infrastructure for geospatial information – should be focused on developing knowledge management and archiving systems as the foundation of the enabling technology for the Grid.

1 Introduction

The ultimate objective of the Alexandria Digital Library Project [1] is to build a global infrastructure for georeferenced information, within the wider context of a global cyberinfrastructure. We emphasize the distinction between *geo-referenced* and *geo-spatial*: geo-spatial information refers to objects which possess a physical location or extent on the surface of the Earth, whereas geo-referenced information refers to any kind of information with which a physical location can be logically associated. For example, a book which describes a place may have the location of that place - usually expressed in latitude and longitude - associated with it.

Access to information by geo-reference is the basis for the paradigm of the *Digital Earth*, a movement given a powerful political impetus in the United States through a speech by Vice-President Gore in 1998:

"I believe we need a *Digital Earth*: a multi-resolution, three-dimensional representation of the planet, into which we can embed vast quantities of geo-referenced data …Earth as it appears from space ….continents, …regions, countries, cities ….land cover, distribution of plant and animal species, real-time weather, roads, political boundaries, and population…. digitized maps overlaid on the surface of the Digital Earth, newsreel footage, oral history, newspapers and other primary sources …[searching] through space, [and traveling] through time" [2].

This vision clearly widens the scope of geo-referenced information to include any information to which a geo-spatial reference can be attached: a guide book, a photograph, a historical event, an archaeological find - a vision which is convincing and intuitive. It speaks to the fact that, far more than we usually realize, our cognitive and reasoning processes are spatially and temporally oriented. It does not, of course, replace more conventional means of classifying and ordering information. But it does offer a very natural, immediate and exciting way of exploring models of the world, as a user-friendly complement to conventional library search techniques.

It is therefore surprising and disappointing that progress in the years since this speech has been so slow. In the US there have been testbeds, focusing mainly on natural disaster scenarios, but these have very much been prototypes of ad hoc construction, and have resulted in little or nothing of any wider and/or longer-term applicability. Even more surprising, the concept of the Digital Earth seems hardly to have caught the imagination of the research community in Europe at all, as a search of EC and national projects shows. In our view, this is because there has so far been too little emphasis on the "tough technical issues" referred to - but not identified - in a subsequent passage of the Gore speech. We believe that the main problem is that there has not yet been a sufficiently coherent attack on the fundamental issues relating to the establishment of a global georeferenced information infrastructure. Nor has there been an attempt to identify those specific aspects which are essential for the support of the Digital Earth concept.

2 The technical challenge

So what are the tough technical issues? First, we believe there is a need to develop a new kind of world wide web, or rather, two parallel webs – the *ephemeral* (here today and gone tomorrow) and the *persistent*. The transience of information on today's ephemeral web is clearly not in itself a negative attribute: it allows information creativity and transfer at a rate previously unimaginable. But in the excitement for this new medium, the importance and value of traditional libraries has tended to be forgotten, or at least devalued. Yet we contend that the traditional social and administrative structures which have been responsible for conserving the world's accumulated knowledge in the past are the natural guardians of the new digital knowledge, in the same way that they inherited responsibility for printed books. So we envision a *persistent* web: a global network of interoperating digital libraries and services, which will become the repository of all knowledge deemed worthy of indefinite preservation. It is often forgotten that the first function of a traditional library is to decide what is worth keeping. So in our vision, knowledge will constantly flow – but through a filter – from the ephemeral to the persistent web.

Second, we observe a strange contradiction between the generally accepted object-oriented way in which at least computer scientists model the world, and the way in which knowledge is stored in libraries. We do not know of any university which holds a library of *active objects* i.e. instances of data structures with associated behaviors.

Software libraries are still considered quite separate from conventional libraries – and very different. And yet, as digital libraries expand the types of objects of which they build collections, it becomes ever more obvious that they should also collect corresponding methods/services which are naturally associated with the objects. Even with an electronic book, there is a need for associated methods: as the simplest example, the blind need a service to read the book to them.

This leads us to the further observation that almost all applications on the web today are *vertically integrated* i.e. composed of (possibly) several layers of services over which neither the user nor the interacting services themselves have any control or choice. There is no concept of a user or an intelligent agent or broker selecting a suite of services themselves from a library of alternatives. This is extremely limiting and inflexible, and the major reason, we believe, why attempts at integrated Digital Earth scenarios have so far been unconvincing – they cannot be reconfigured 'on the fly' to new scenarios.

The central technical objective of our vision, therefore, is to build a framework for a global persistent web of digital libraries of active objects and flexible, open services which are both vertically and horizontally interoperable, and are themselves stored in the libraries as object methods.

3 A four-layer model of the persistent web

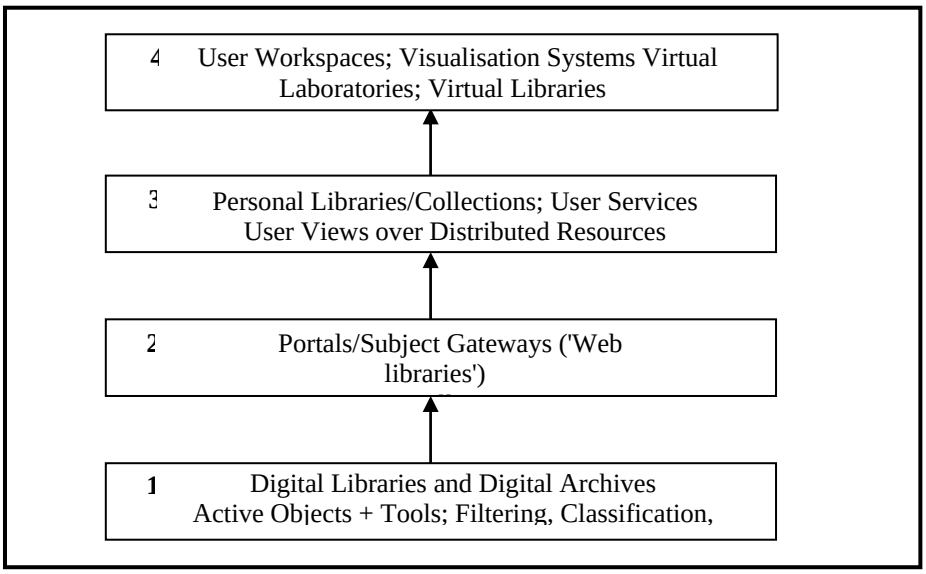


Figure 1: A four-layer model for a global information infrastructure

Our overall conception of the framework for a global information infrastructure is represented in the four-layer model illustrated in Figure 1. This model is an attempt to partition *functionality* - it does not imply anything about how the execution of that functionality may be physically or logically distributed. It does however begin to develop interface levels at which interoperability standards may be defined.

The foundation of this framework is the *digital library* and the *digital archive*. The term 'digital library' is now loosely used to describe any web application which serves information to users. For us, however, a digital library must be first and foremost a *library*, i.e. it must be a *managed* collection of information and services with all the functions of a traditional library: selection policies, classification systems (subject indexing), catalogues, metadata (content description) standards, and user services such as searching, delivery and helpdesk functions [3,4,5].

We therefore emphasize the importance of the digital *archive*, as a library which has the responsibility for the persistent storage and physical preservation of its digital collections. In our view, this responsibility will be taken in future primarily by national, academic and public libraries, as a natural yet challenging extension of the role which they have always taken. We foresee the development of two parallel and interconnected worlds on the web: the one - which we have today - facilitates the rapid exchange of new information and ideas. The other - the global knowledge infrastructure - will hold all accumulated knowledge distilled from this invaluable but ephemeral information exchange. At present, unfortunately, this distinction is not clear, because no formal knowledge infrastructure exists, and the library world has not yet fully risen to the challenge which it faces.

In our overall structure, we envisage that subject gateways (level 2) will provide indexes to both aspects of the web, i.e. to ephemeral web sites and to permanent archives at level 1. Level 3 contains user services, i.e. software which analyses and processes the archived data sources in some way. (Note, however, that these services will be *stored* in an archive at level 1). Level 4 is the user presentation level - primarily visualization.

The key technology supporting a georeferenced DL infrastructure at all levels is a database system which includes spatial operators and a subset of Geographic Information System (GIS) functionalities. This geospatial functionality, however, must be combined with text and date information retrieval and special services such as *digital gazetteers* designed for DL and information retrieval purposes [6,7]. This combination of functionality greatly expands the types of information retrieval and evaluation that can be supported, some of which are completely beyond the capabilities of conventional database and information retrieval systems.

At the archive and portal levels and the personal libraries at level 3, the database system needs to support the metadata models associated with the object types in each library collection. These may be based on the metadata standards designed for geospatial information (e.g., the FGDC metadata standard or its ISO equivalent) [8] or on the MARC standard used for library cataloging. An added challenge is to represent information objects, datasets, modeling and simulation software, and other services in such a way that programmatic interfaces can be designed to permit on-the-

fly identification and utilization of DL objects for visualizations and simulations and other manipulations that cross the four-levels of the conceptual framework.

4 A geo-spatial illustration

As a geo-spatial example, an archive might contain aerial photographs of a particular region. The user wants to 'fly through' this area in a full virtual reality environment, or at least through an interactive and dynamic visual display. (S)he must first invoke a user service (from an archive) to perform this function. The service may consult a gateway at level 2 to select one of several archives which contain aerial photography of the specified region, or may combine the contents of several archives. Or the user may be offered a choice of archives, with metadata provided to aid in the execution of that choice. The service then dynamically transforms a subset of the archive data into an appropriate form to supply to the visualization environment at level 4. If access to the archives is very time-consuming, or if the user buys the data, then (s)he may wish to archive it locally for subsequent use. There must therefore be a service at level 3 supporting a personal library (for which the term 'derived' library has recently been coined) available to the user environment at level 4.

To appreciate the nature of the challenge as we see it, and the magnitude of it, we have only to consider a conventional approach to this 'fly-through' example. Without a library of portable software, the user has somehow to locate, acquire and install a fly-through program. And it is almost certain that, in order to facilitate the interactive display of the fly-through in real time, this program will require data sets in a specific pre-defined format.

This is a closed system with no choices. In contrast, we want to create an open system whereby the user in this example can first select a fly-through program (a level 3 service), with known functionality, from an archive (level 1). When the user runs this program and selects a fly-through region, the program itself will send out a request to a software agent (level 2) to locate an archive source for the requested fly-through imagery or maps. In all probability the archive data will not be in the appropriate format, and a conversion service - or a sequence of services - will have to be invoked *automatically* to perform the necessary transformations. These services must also be identified and retrieved from a level 1 library.

The challenge is to make this work, and to make it work in real time. Clearly this requires high-speed access to - and high-speed processing of - large data sets. *But it is the move from a closed to an open computing environment which presents the really new challenge*s. For example, at the service and agent levels, how is the fly-through program to specify precisely the type of imagery or map which it needs? And how is the agent to determine what sequence of conversions might be needed, and the location(s) of suitable conversion software, and the type of the ultimate target source data in an archive?

We believe that most of the tools and techniques needed to answer these questions and solve these problems already exist: scripting languages; ontologies; description logics; MIME types; self-describing documents (XML) and files; metadata standards; object-oriented programming languages and database systems; web search engines;

web crawlers; software agents; and open hypermedia systems. The challenge is to construct a coherent georeferenced information infrastructure from these components.

To meet this challenge, we propose to base the foundation layer of the infrastructure on components already developed within the Alexandria (geo-referenced) Digital Library; the persistent storage technology (SRB) from San Diego Supercomputer Center [9], and the Globus grid toolkit [10]. Upon this foundation, the higher-level services of the four-layer model, including the key horizontal and vertical service brokerage mechanisms and visualization systems, can be constructed.

5 Relationship to The Grid

Since the early 60's, if not earlier, the database community has dreamed of a network of independent, heterogeneous computer systems offering interoperable data and services. Semantic interoperability has remained an elusive goal, but the extraordinarily rapid rise and success of the World Wide Web has demonstrated how much can be achieved in that direction with what is really a very simple set of well-designed open standards. For all that, the Web is still essentially a passive medium: we have not yet realized the long-held vision of user-specified tasks being resolved by intelligent software agents into a sequence of services assembled and executed within a distributed environment of data and software resources.

However, the Grid community, which is still widely dismissed within the DL community as interested only in setting up computer networks for massively parallel computations, is in fact evolving rapidly towards this vision of a Web of active and coordinated services, and is making rapid advances in the development of interoperability standards and protocols. Further, recent papers in the field clearly recognize the need for metadata repositories describing data and software archives. It is true that the application focus in this community is still mainly restricted to large scientific data sets and associated data analysis services, but this is an area which the DL community has ignored almost entirely.

It is extraordinary and regrettable that almost no DL research is being devoted to the question of how to integrate DLs into this developing infrastructure, despite the fact that the whole future of digital libraries, and perhaps of all libraries, depends on the answer to this question. It is to be hoped that the recent *cyberinfrastructure* report in the US [10], which is seen as the basis of a major new research initiative by the National Science Foundation, will encourage research in this direction. But, while it envisions the development of a multi-layer infrastructure very similar to that proposed above, there is a danger that its emphasis on science and engineering applications may reinforce rather than reconcile the division between the Grid and DL communities.

6 The way forward

We nevertheless argue that future DL research should focus on integration within the cyberinfrastructure framework being developed by the Grid community, and on the development of DLs of active encapsulated objects which include large data sets and associated analysis tools. Only in this way can we ensure that DLs, and the essential

expertise of the library community, are properly represented and integrated in future generations of Web technology.

We also see the need for the convergence of research efforts which have been conducted over many years in a number of disciplines – not least in library science – in the area of knowledge organization systems (KOSs) such as thesauri, ontologies and concept maps, and in the area of georeferenced information – in particular, gazetteers. We see these KOSs as essential adjuncts to a DL: not library collections in themselves, but capturing semantic and conceptual knowledge to aid in information search and interpretation of library information. Specifically, we aim to use KOS tools to advance from keyword-based search to concept-based search, in conjunction with georeferenced and temporal search.

We also aim to restore the original Digital Earth vision**.** In the intervening years since the Gore speech, the Digital Earth concept has unfortunately acquired a much more restricted meaning. It has been adopted by the Earth Sciences community to describe their ultimate vision of capturing a complete digital model of the entire planet – or at least of capturing a geospatial and temporal model of constantly increasing detail and accuracy. This conception is driven mainly by research to model physical phenomena on a global scale, such as weather forecasting, climate change modeling and environmental monitoring, and is seen as the ultimate GIS system. Our objective, however, is to restore the original vision by focusing on logical georeferencing of cultural knowledge. A key aspect of this research is the attempt to unify geospatial and 'conventional' (i.e. text-based) DLs, by introducing geospatial search as a complementary library search paradigm. A major challenge is the development of efficient and accurate technology for automated georeferencing of (digitized) text documents. In this context, a new generation of gazetteers is assuming a major role.

A full instantiation of our conception of the Digital Earth would, however, need to subsume the narrower geospatial concept, and this raises another major and exciting challenge in the development of georeferenced DLs - a challenge which clearly illustrates the overall need to integrate DLs within an active Grid computing framework: can we implement GIS functionality as a set of distributed services on geospatial DL collections within the cyberinfrastructure?

References

1. *The Alexandria Digital Library*: Publications, research papers, current bibliography. http://www.alexandria.ucsb.edu.

2. A. Gore. *The Digital Earth: Understanding our planet in the 21st century*. http://www.opengis.org/info/pubaffairs/ALGORE.html.

3. S. Harter. *What is a Digital Library? Definitions, Content and Issues*. Int. Conf. on Digital Libraries and Information Services for the 21st Century, Seoul, South Korea, September 10-13, 1996. http://php.indiana.edu/~harter/korea-paper.htm

4. M. Collier. *Towards a general theory of the Digital Library*. Proc. Int. Symp. on Research, Development and Practice in Digital Libraries (ISDL97), Ibaraki, Japan, November 18-27, 1997. http://www.dl.ulis.ac.jp/ISDL97/proceedings/collier.html

5. W. Arms, *Digital Libraries.* Pub. MIT press, December 1999. ISBN 0-262-01180-8

6. L. Hill. *ADL gazetteer content standard*. http://www.alexandria.ucsb.edu/ lhill/aleximp/gaz content standard.html.

7. L. Hill. *Thesaurus of geographic feature type terminology*. http://www.alexandria.ucsb.edu/ lhill/html/index.htm.

8. Federal Geographic Data Committee (FGDC). *Geospatial data clearinghouse activity*. http://fgdc.er.usgs.gov/Clearinghouse/Clearinghouse.html.

9. A. Rajasekar, M. Wan, R. Moore, W. Schroeder, G. Kremenek, A. Jagatheesan, C. Cowart, Bing Zhu, Sheau-Yen Chen, R. Olschanowsky. *Storage Resource Broker - Managing Distributed Data in a Grid*. Submitted to Computer Society of India Journal, special issue on SAN, 2003. http://www.npaci.edu/DICE/Pubs/CSI-paper-sent.doc

10. *The Globus Toolkit* http://www.globus.org

11. *The Atkins Report* http://www.cise.nsf.gov/sci/reports/toc.cfm

Towards a Service-oriented Architecture for Collaborative Management of Heterogeneous Cultural Resources

Jérôme Godard[†⋆], Frédéric Andrès[‡], Elham Andaroodi[†], and Katsumi Maruyama[‡]

National Institute of Informatics, The Graduate School
Hitotsubashi 2-1-2, Chiyoda-ku, Tokyo 101-8430, Japan
& SOKENDAI - The Graduate University of Advanced Studies
Shonan Village, Hayama, Kanagawa 240-0193, Japan
[†]{jerome,elham}@grad.nii.ac.jp, [‡]{andres,maruyama}@nii.ac.jp

Abstract. Managing cultural knowledge for collaborative projects generates many issues. We are facing them as we build a collaborative digital archive related to the historical silk roads. We present here the architecture we defined in order to support highly relevant resource management and to provide adaptive services.

1 Introduction

This paper gives an overview of a generic architecture we are currently building as part of the the Digital Silk Roads project (DSR [11]). Initiated by UNESCO and NII, DSR aims at providing a wide area collaborative portal for research and education in order to collect, to archive, to organize and to disseminate all the *relevant* information that can be gathered about the historical silk roads. This implies to deal with any kind of multilingual digital document; it goes from architecture-related pictures showing parts of buildings to traditional songs that characterize some social behavior. Therefore, DSR wants to provide adaptive services to users, depending on all the knowledge we have on the *environment* (user himself, communities he's involved in, and device he's using); we are dealing with more than 400 experts in many fields providing annotated *resources* (i.e. monotype multimedia digital documents) and the set of end users is indefinite. This vision requires to define an advanced model for the classification, the evaluation, and the distribution of multilingual multidisciplinary cultural *resources*. Our approach fully relies on state of the art knowledge management strategies. We define a global collaborative architecture that allows us to handle resources from the gathering to the dissemination.

In the following section, we introduce our testbed project and its portal. Then, after showing our interest in ontology, we present our information management model in section 3. Based on this model, the fourth section describes and define the personalized services we are providing for collaborative environments. The last section will give some forthcoming issues.

⋆ This research is partially supported by a grant (*bourse Lavoisier*) from the French Ministry of Foreign Affairs (*Ministère des Affaires Etrangères*).

2 DSR Framework

2.1 Description

We are involved in the Digital Silk Roads project (DSR [11]), which is focused on the collaborative management of digital multilingual cultural documents; it is handling all kinds of multimedia documents (including text-based, image, audio, video types) related to the historical silk roads. DSR aims at creating a global repository that enables us to collect, validate, preserve, classify and disseminate cultural resources. Building such a system is a great challenge, and requires to fulfill some commitments: first, it must provide an appropriate knowledge management framework that supports all the tasks it aims at covering. Then, documents and annotations have to be gathered and to be well structured. Thirdly, the data has to be stored safely. Afterwards, it is necessary to ensure an accurate access to the resources for each user. Finally, the distribution of the information has to be optimized in order to propose adaptive services. The global framework of DSR with its data distribution scheme is illustrated on Fig.1.

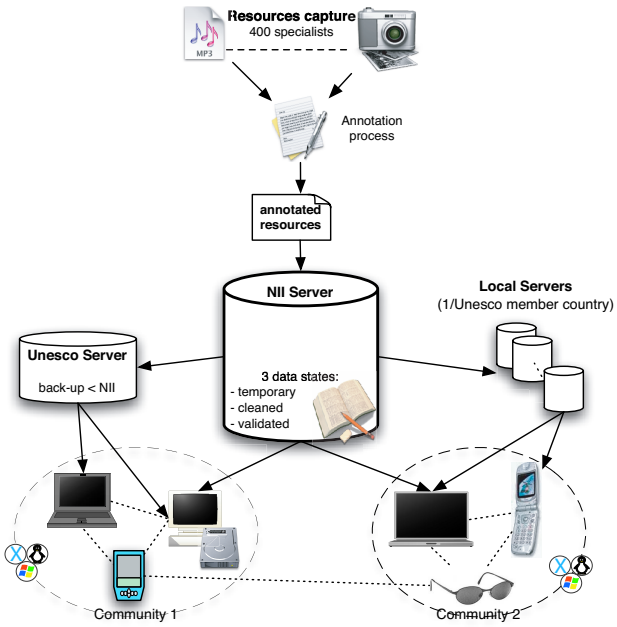


Fig. 1. DSR framework

The main issue we are facing with DSR is heterogeneity; we are considering very varied data (as said before), users and devices. DSR users are divided into two categories: contributors and end users who are supposed to manipulate any kind of device

(from mobile phones to mainframe computers). Since DSR is a collaborative project, it is important to register the users as members of communities (NB each community has an *access point*, which is a device being a kind of sub-server dedicated to the community); this enables us to increase the environmental knowledge that is required for performing adaptive services based on users' status and abilities.

2.2 ASPICO Platform, an Open Archives Initiative based Digital Archive

Several research projects such as the arXiv e-print archive[1], the Networked Computer Science Technical Reference Library (NCSTRL)[2] or the Kepler project [9] in the field of digital libraries or digital research archives, tried to solve issues of sharing research information. They generally provide a common interface to the technical report collections based on the Open Archives Initiative (OAI) infrastructure[3]. This mechanism enables interoperability among large scale distributed digital archives. In many cases, the network environment services include automated registration service, tracking of connected clients, and harvesting service of clients' metadata. Query service enables accesses to resources and to its related metadata. OAI has created a protocol (Open Archives Initiative Protocol for Metadata Harvesting, OAI-PMH) based on the standard technologies HTTP and XML as well as the Dublin Core metadata scheme[4]. OAI presently supports the multipurpose resource description standard Dublin Core which is simple to use and versatile. Shortcomings of such research projects generally include a too general metadata attributes schema for fine-grained information (e.g. cultural domains) and the non-support of community building. However, OAI-PMH itself has been created to provide an XML-wrapper for metadata exchange. It has been extended in the Digital Silk Roads project to support multi-disciplinary metadata schemas such as Object ID[5] for historical buildings, Categories for the Description of Works of Art (CDWA) for historical artifacts, or VRA[6] for visual resources. In order to avoid the different shortcomings and to provide a community framework for the research and education on Digital Silk Roads, we proposed and built the Advanced Scientific Portal for International COoperations on Digital Silk Roads platform[7] (ASPICO-DSR). ASPICO-DSR is OAI-PMH 2.0 compliant as part of the distributed collaborative architecture as it is shown in Fig.2. The platform provides services for data handling, registration for identification, and metadata handling based on cross-disciplinary metadata schemas to create OAI-compliant metadata and resource management. Researchers can annotate resources according to their point of views and can share their comments according to cross-disciplinary and multi cultural backgrounds. Furthermore, the cultural resource server includes an ontology management service to support multi-lingual ontologies of cross-disciplinary metadata standards and multi-lingual ontologies in Digital

[1] arXiv.org e-Print archive: http://arxiv.org/
[2] Networked Computer Science Technical Reference Library (NCSTRL): http://www.ncstrl.org/
[3] Open Archives Initiative http://www.openarchives.org/
[4] Dublin Core: http://dublincore.org/
[5] Object ID http://www.object-id.com/
[6] Visual Resources Association: http://www.vraweb.org/
[7] ASPICO-DSR: http://aspico-dsr.org

Silk Roads related fields (e.g. architecture, history, geography, art. . .). We currently use Protégé 2000[8] as the Ontology manager. The start point of the ASPICO storage management has been the Dspace[9]. We also have extended Dspace core system to produce a multi-lingual platform and to support DSR metadata.

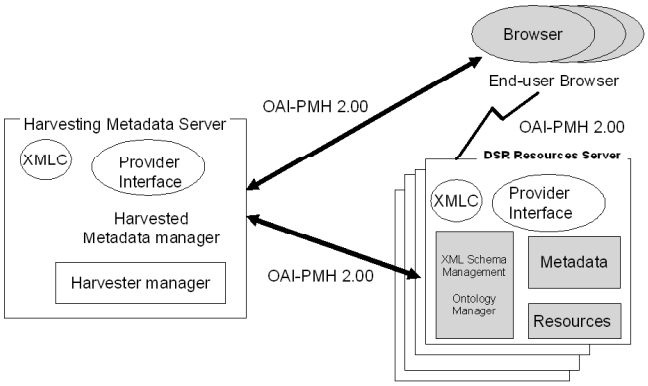


Fig. 2. ASPICO architecture

3 Knowledge Management

3.1 Handling Ontologies

DSR's ASPICO portal has to deal with large repository of multimedia of historical and cultural resources which come along with the web. For instance, it provides access to databases containing cultural heritage photography. Meanwhile semantic understanding, access and usages of these materials are not fully possible due to the semantic gap for their annotation and retrieval [12]. There are still shortcomings of appropriate methods and tools for multimedia annotation, browsing and retrieval to help the users to find what they are really looking for. On the other hand, historical and cultural content of these databases make the process more complicated as there might be different semantic interpretations toward the subject of the visual information. Development and application of multi-lingual multimedia ontologies for the conceptual recognition of the content of cultural heritages of silk roads by using domain knowledge is the approach of ASPICO to improve multimedia semantic annotation and retrieval. Ontology is defined as a specification of a conceptualization or as a set of concept-definition, a representational vocabulary[10]. Another definition of ontology which emphasizes the

[8] http://protege.stanford.edu/
[9] DSpace Federation http://www.dspace.org/
[10] Gruber-Tom, "What is an ontology?" http://www-ksl.stanford.edu/kst/what-is-an-ontology.html

component-base recognition of a subject is a declarative model of the terms and relationships in a domain or, the theory of objects[11]. Based on these definitions ontology provides a hierarchical structured terminology of a domain and is completed by defining different relationships between term-sets. Ontology is explicitly declared to be helpful for knowledge representation, knowledge sharing and integration, portability issues... Ontology has application in artificial intelligence, natural language processing, multimedia database... Examples of ontology can widely be found in Biomedicine. Meanwhile recently in the field of multimedia enhanced annotation and retrieval, like photo annotation and ontology-based image retrieval and in some cases with relation to cultural heritage and art objects ontologies are designed and applied. In the field of cultural heritage, through application of domain ontology, the domain experts can develop semantic annotation for the multimedia data like images, and users can have access to a model of the vocabularies of the subject which will guide them for a more standard and intelligent search through database and will lead to a better retrieval.

DSR is directly involved in servicing ontology management on a case study of architectural cultural heritage named caravanserais[12]. This ontology tries to design a visual lexical model of terms or components in architectural relic and relationship between components based on the physical and spatial characteristics of the components. It also tries to design the ontology in different languages with the help of UNESCO expert team in order to exchange the content with experts and cover the needs of multilingual users [2]. This ontology, which is designed with Protégé 2000 environment (version 2.1) will be accesible as part of ASPICO and will be used by domain experts in order to reach a consensus for its content to be extended to other languages and typologies of architectural heritage. Developing ontology on this case study as part of the portal is considered as a proper example for involvement of domain experts over internet in knowledge management and application of it can help enhanced access to large visual data which DSR is dealing with.

3.2 *IMAM*, an Information Modeling for Adaptive Management

We have the great opportunity for DSR to be working with more than 400 specialists in various fields (using 21 languages) who are *motivated* and *able* to annotate documents very accurately. These annotations become very valuable once they are related to the knowledge structure presented above. Then, we need a unified model that enables us to capture this useful information about the documents and also the available knowledge about communities, users and devices.

The first requirement that appears to us when aiming at defining a generic model is to fit standards as much as possible and to adopt efficient technologies (here comes XML). Then it is necessary to be very rigorous and to describe very precisely the whole data structure; an information model must be based on a coherent algebraic model that

[11] Roberto Poli, "Framing Ontology - Second Part" http://www.formalontology.it/Framing_second.htm

[12] This multi-lingual ontology on architecture is constructed as part of a colaboration between National Institute of Informatics in Japan and the Architecture school of Paris Val de Seine in France under the *Digital Silk Roads Initiative Framework* in cooperation with UNESCO.

can support all the layers of the architecture (physical, logical, semantical, transactional). In our case, it has to handle physical entities (servers, *access points*, normal devices), knowledge management entities (*resource's description*, community's, user's, and device's *profiles*) and semantic entities (types of documents, i.e. *categories*, attributes, i.e. *descriptors*, and characteristics, i.e. *descriptors' values*)[6].

In order to categorize and describe *resources* (i.e. any mono-type multimedia document that can be related to at least one topic), we use an ontology-like knowledge tree called *Resource Categorization Tree* (RCT,); a node of the RCT is denoted α_i. The knowledge management through the RCT is based on a contextual structure. Our model aims at describing as clearly as possible the information contained in the annotations provided on the *resources*. The primary element in this approach is the *descriptor* (δ), which is a contextual attribute (dimension). The values assigned to the jth *descriptor* of the ith node for a specific *resource* are denoted $\sigma_{i,j,k}$. The whole annotation about a *resource* is contained in a complete branch of the RCT called *Resource Description* and defined as follows: $D_r = \left(<\alpha_i>, <\delta_{i,j}>, <\sigma_{i,j,k}> \right)_{\substack{i=0,\ldots,m-1 \\ j=1,\ldots,p \\ k=1,\ldots,q}}$. The information about communities, users, and devices[13] is contained in a quite similar structure called *profile* (denoted π) that is composed of *descriptors* and *descriptors' values* (without nodes): $\pi = (<\delta_v>, <\sigma_{v,w}>)$. We specified DBMS-like operators [5] with the aim to powerfully manage the *resources* (e.g. *create, insert, edit, intersection, difference*...) by using *IMAM*.

4 *IMAM*'s Services

4.1 Preamble

Services to be proposed to DSR users cover usual database functions, personalized automated processes, and management of transactions in order to ensure the capability of the system to work in heterogeneous distributed mobile environments. In fact, more and more people are showing strong interests in peer-to-peer [1] as a foundation for creating advanced distributed applications; moreover, innovative sharing strategies are implemented and used in peer-to-peer [13, 10, 3] and mobile systems [8, 7]. But they are generally lacking in a unique generic basis for knowledge management that would allow us taking fully advantage of these powerful distributive environments. We agree that distributed knowledge management has to assume two principles [4] related to the classification: autonomy of classification for each knowledge management unit (such as community), and coordination of these units in order to ensure a global consistency. Distributed adaptive services require to exploit all the environmental knowledge that is available about the elements involved. An important category of this knowledge is related to devices' states; indeed, knowing if a device is on, in sleep mode, off, if its battery still has an autonomy of five minutes or four days, or if it has a wired or wireless connection, etc. helps adapting services that can be delivered to this device. For each device, we consider a state control that is part of the device's *profile*. And of course

[13] NB: a user can be involved in several communities and a device can be shared by different users being involved in different communities.

we use the information contained in communities' and users' *profiles*. The information that can be gathered in collaborative environments (i.e. people sharing interests and resources) shall increase the ability to create new kinds of services. Personalized services finally depend on user-related contexts such as localization, birth date, languages abilities, professional activities, hobbies, communities' involvement, etc. that give clues to the system about users' expectations and abilities. All this information is quite easy to extract and to manipulate through *IMAM*; in the remainder of this section, we present the two main adaptive services based on our model.

4.2 Authoritarian Data Placement

The main motivation for the data placement is to automatically copy resources that seems to be very relevant to a user or a community on the appropriate devices. This operator relies on memory spaces that are allocated on each device for the server to place the data. Each time a *resource* is added to the server, its *Resource Description* is used for analyzing the possible correlations with the communities and users interests. The strategy we are using to evaluate the significance of a *resource* placement on a device is quite similar to the one used for operator *sim* (which evaluates the similarity between two *resources*, see [5]). But in the case of the placement (operator denoted *disp*), the *descriptors* are replaced by the *descriptors' values*. we extract the ratio of common *descriptors values* where the *descriptors* are similar by using the function ρ_D: $\rho_D(A,B) = \frac{\mid \text{TINTER}(A,B) \mid}{\mid \text{TUNION}(A,B) \mid} \in [0,1]$, with:

- $\text{TINTER}(A,B) = \{<\sigma_{inter}> \mid \sigma_{inter} = \sigma_{i,j} = \sigma_{k,l}, (\sigma_{i,j} \in A) \wedge (\sigma_{k,l} \in B) \wedge (\delta_i = \delta_k)\}$
- $\text{TUNION}(A,B) = \{<\sigma_{union}> \mid (\sigma_{union} \in A) \veebar (\sigma_{union} \in B)\}$

where A and B contain ordered families of labels, which are lists of descriptors with associated values (there can be only one label, in the case of a profile for instance). $\veebar$ denotes operator *exclusive-or*.

The *disp* operator first applies ρ_D to communities. Depending on two threshold values s_{c_1} and s_{c_2} ($s_{c_1} > s_{c_2}$), we decide if the resource has to be placed on all the devices used in the community (Case 1 on Fig.5) or only on the community's *access point* (Case 2 on Fig.5); the operator dispatches the *resource* on all devices of a community for which the value returned by ρ_D is higher than s_{c_1}, and if the *resource* seems to be quite relevant only for a community (i.e. the returned value is between s_{c_1} and s_{c_2}), the operator copies the *resource* on the *access point* only. The last option for *disp*, when the *resource* does not seem to be relevant for a whole community (Case 3 on Fig.5), is to apply ρ_D on each user in this community; again, this is done by using a threshold value s_u. If the value returned by the function is higher than s_u, then the resource is placed on the user's device that is the most able to get it. The selection of the device is processed by the function $\text{SELECTDEV}(i,j)$ i and j being integers, the function returns the device (profile) used by the j^{th} member of the i^{th} community that has the largest storage capacity on its placement area (see Fig.3).

We have to mention that each time a *resource* is supposed to be placed on a device, *disp* first checks the ability of the device to store the *resource* and if there is not enough

```
SELECTDEV(i, j)
1  device ← ∅
2  a ← 0
3  for k ← 1 to K_{i,j}
4       do if FSPACE(π_{d_{i,j,k}}) > a and STATE(d_{i,j,k}) = TRUE
5            then a ← FSPACE(π_{d_{i,j,k}})
6                 device ← π_{d_{i,j,k}}
7  return device
```

Fig. 3. The device selection function pseudo-algorithm

space for it, the operator compares the new *resource* to the *less interesting resource* that is on the placement area of the device. If the new *resource* is more *interesting*, then it shall replace the other one. This is recursively done by the function PROEMIN(D, π, ρ), D being a resource description, π a device profile, and ρ a value between 0 and 1 (see Fig.4).

```
PROEMIN(D, π, ρ)
 1  proemin ← TRUE
 2  if STATE(π) = TRUE
 3     then if ASPACE(π) > SIZE(D)
 4             then PUT(D, π)
 5             else  t ← GETWR(π)
 6                   if ρ > t[1, 1] and FSPACE(π) > SIZE(D)
 7                     then DELETE(t[1, 2], π)
 8                          PROEMIN(D, π)
 9     else  proemin ← FALSE
10  return proemin
```

Fig. 4. The Proemin function pseudo-algorithm

Before copying a *resource* on a device, *disp* checks if the device is online and if it has enough free space on its placement area (limited predefined space) for the *resource* to be stored. The storage capacity (full capacity and empty space) of a device is defined in order to ensure limits (depending on a minimum and a ratio) that cannot be passed over; the available space dedicated to automated services on the device must be precisely defined (default ratio or user's choice) in order to keep enough memory space for the user's *manual* activities. *disp* gets this states' information from the device *profile* via several functions:

- FSPACE(π_d) returns the full space allocated for placed data on the device d (in KB).
- ASPACE(π_d) returns the available space in the placement area on d (in KB).

- SIZE(D_{r_i}) returns the size of Resource r_i (in KB).
- STATE(π_d) returns FALSE if the device d is off, and TRUE if it is on.

Thus appears the update problem: variables we need to handle can change at any time very irregularly (frequency might anyway be taken into account); for instance, it is necessary to record the new locations of the *resource* in its *resource description* and devices' states. Indeed, to be able taking advantage of the dispatched *resources* for the query management, we have to keep a record of all the locations a *resource* is stored at. So each PUT and DELETE (see below) implies that the *Resource Description* (which contains all these locations within the *locations descriptor*) is updated. The new version of the *Resource Description* is first saved on the server, and then it overwrites the other copies that are on the devices containing the *resource*. The updates processes have to take into account the possibility for a device to be offline, and so to ensure that the update can be performed as soon as the device becomes available. Following the same strategy, when a device is switched on, it updates its IP address in its *profile*, which is copied on the server and related *access points* (we do not address here the case of connection loss because of space limitation). We also have to consider the creation of new communities: each time a community is created, the placement operator must be applied on the server to check what *resources* should be dispatched on the devices of this community. The function UPDATEPROF() provides the support described above for every *Resource Description* and *profile* that has to be updated.

We finally declare all the functions that *disp* uses in order to manipulate the *resources* and their *profiles*:

- PUT(r, d) accesses the placement area on the device d and pastes the Resource r there.
- GETPROFCOM(x) (x being the number (*i*) of the i^{th}community, or the community's identifier *comID*) returns the profile of the related community. GETPROFUSE(x) works the same way for a user.
- GETAC(π_c) returns the profile of the *Access Point* of the community π_c.
- DELETE(x, π_d) deletes the resource identified by x on the placement area of the device d.
- Each device's profile contains a table $[r_i, \rho_i]_{i=1...n}$ made of n columns (n being the number of resources stored on the device) and two rows (resource identifier and related ρ_D values) such as ρ_D values are increasingly ordered. The function GETWR(π_d) returns this table for the device d.
- SELECTDEV(i, j) i and j being integers, the function returns the device (profile) used by the j^{th} member of the i^{th} community that has the largest storage capacity on its placement area (see Fig.3).

NB: some variables are shared and are accessible from all the functions dedicated to the services; it consists in all the profiles (communities ($< \pi_{c_j} >_{j=1,...,\mathcal{C}}$), users ($< \pi_{u_{j,k}} >_{k=1,...,\mathcal{U}_j}$), and devices ($< \pi_{d_{j,k,l}} >_{l=1,...,\mathcal{L}_{j,k}}$)), sets' number of elements ($\mathcal{C}$ is the total number of communities, $\mathcal{U}_j$ is the total number of users involved in the j^{th} community, $\mathcal{K}_{i,j}$ is the total number of devices used by the j^{th} user of the i^{th} community), and threshold values (s_{c_1}, s_{c_2}, s_u).

```
DISP(D_r)
 1  disp ← FALSE
 2  for i ← 1 to C
 3       do π_{c_i} ← GETPROFCOM(i)
 4          ρ_1 ← ρ_D(D_r, π_{c_i})
 5          device1 ← GETAC(π_{c_i})
 6          if ρ_1 ≥ s_{c_1}
 7            then for j ← 1 to U_i                                   ▷ Case 1
 8                      do π_{u_{i,j}} ← GETPROFUSE(i, j)             ▷ Case 1
 9                         device2 ← SELECTDEV(i, j)                  ▷ Case 1
10                         if PROEMIN(D_r, device2, ρ_1) = TRUE       ▷ Case 1
11                           then UPDATEPROF()                        ▷ Case 1
12                                disp ← TRUE                         ▷ Case 1
13          elseif s_{c_2} ≤ ρ_1 < s_{c_1} and PROEMIN(D_r, device1, ρ_1) = TRUE   ▷ Case 2
14            then UPDATEPROF()                                      ▷ Case 2
15                 disp ← TRUE                                       ▷ Case 2
16          else for j ← 1 to U_i                                     ▷ Case 3
17                      do π_{u_{i,j}} ← GETPROFUSE(i, j)             ▷ Case 3
18                         ρ_2 ← ρ_D(D_r, π_{u_{i_j}})                ▷ Case 3
19                         device3 ← SELECTDEV(i, j)                  ▷ Case 3
20                         if ρ_2 ≥ s_u and device3 ≠ ∅               ▷ Case 3
                           and PROEMIN(D_r, device3, ρ_2) = TRUE      ▷ Case 3
21                           then UPDATEPROF()                        ▷ Case 3
22                                disp ← TRUE                         ▷ Case 3
23  return disp
```

Fig. 5. The placement pseudo-algorithm

The *disp* placement operator has been introduced in [5]; we proposed here a full description of the pseudo-algorithm (see Fig.5) that moreover takes into account new features such as checking devices activity and storage capacity.

4.3 Adaptive Query Management

A major benefit of the RCT is to allow us giving an appropriate *viewpoint* (denoted ν) to each user for a same set of *resources* (taking the user's characteristics and environment into account). In fact, our *viewpoint* can be seen as a query optimizer, since it clears and modifies an initial set of *resources*. It is defined as follows [5]:

$$\nu = \xi \circ \psi : \mathcal{R}^p \times \Delta^p \times \Sigma^p \times \Pi \longrightarrow \mathcal{R}^q \times \Delta^q \times \Sigma^q$$
$$\left(<D_i>_{i=1,\ldots,p},\, \pi_e\right) \overset{\Xi \circ \Psi}{\longmapsto} <D_k>_{k=1,\ldots,q}$$

where p is the number of considered *Resource Descriptions* and q the number of returned *Resource Descriptions* ($q \leq p$), π_e is the *profile* of the *environment e* (with $\pi_e = \pi_u \cup \pi_d$, u denotes a user and d a device), and Ξ and Ψ are two sets of rules: Ξ contains acceptation rules; if a *descriptor value* of the *Resource Description* r (denoted

D_r) does not respect a rule in Ξ, then the set returned by ξ does not contain D_r. Ψ contains transformation rules; if a *descriptor* of D_r is involved in any rule of Ψ, its value might be modified depending on π_e. Each rule is a test on a pair of *descriptor values*; one from the *Resource Description* and one from the *profile*.
Our query optimization strategy requires a large CPU contribution from the servers and devices as they have to apply the *viewpoint* on all the *Resource Descriptions* they are receiving from other devices. However, since we are dealing with high resolution multimedia *resources* and as we are reasonably convinced that portable devices processing capacity will very soon become very high, we claim that the overload on the devices CPU worths the *resources*' selection cost.

The interesting features we are getting from the *disp* operator and the *viewpoint* can then be enhanced by using an appropriate query management based on our three-layers architecture (server, *access point*, normal device). We are currently designing a distributed query manager based on JXTA and BitTorrent[14] (P2P delivery system); in fact *resource descriptions* can partially seen as BitTorrent *trackers*, as they contain all the locations of the *resources*. We now just have to take advantage of *IMAM*'s support to provide appropriate *resources* to users in the best conditions. Following BitTorrent strategy, we can provide distributed query processing by using the placed and indexed data; then a device can access all copies of a *resource* (even not complete ones).

A simple example of what we want achieve with *IMAM*: let us consider a class studying caravans in Iraq with a focus on a the 14^{th} century and looking for people exchanging specific products. It would be interesting and useful for the students to get on their laptop maps, pictures, videos that are related to their topic. This could be done by creating the community some time before the class starts this lesson. The placement would be then restricted by the viewpoint, being used as a filter for the sets of resources to be sent on each device. Then the distribution should be improved (regarding time and bandwidth consumption) by a BitTorrent-like community P2P shared access on the resources. Moreover, a unified protocol based on JXTA can enable the whole process to be more efficient and safer.

5 Conclusion

The framework described in this document corresponds to a long-term vision. Indeed, it requires much time to define the knowledge management structure (i.e. multilingual ontologies and interrelationships) and to implement the portal that has to gather all the information and the system that provides the services. At least, we now have a solid basis for the management of the resources that allow us to design innovative services. However, many issues are remaining; we need to implement the transaction and query managers in order to enable users to start using all the portal functionalities. Then, we have to define some policies to evaluate the relevance and efficiency of adaptive services (e.g. how to fix the threshold values). We also would like to point out that the significance of this architecture is not restricted to collaborative cultural projects; we see valuable possible application in the management of companies' resources.

[14] http://bitconjurer.org/BitTorrent/

References

1. Serge Abiteboul. Managing an XML Warehouse in a P2P Context. In *Proc. of CAiSE*, pages 4–13, Klagenfurt, Austria, June 16-18 2003.
2. Elham Andaroodi, Frédéric Andrès, Kinji Ono, and Pierre Lebigre. Ontology for caravanserais of Silk Roads: Needs, Processes, Constraints. In *Proc. of the Nara Symposium for Digital Silk Roads*, pages 361–367, Nara, Japan, December 10-12 2003.
3. Neal Arthorne, Barbak Esfandiari, and Aloke Mukherjee. U-P2P: A Peer-to-Peer Framework for Universal Resource Sharing and Discovery. In *Proc. of the FREENIX Track: USENIX Annual Technical Conference*, pages 29–38, San Antonio, Texas, USA, June 9-14 2003.
4. Matteo Bonifacio, Paolo Bouquet, and Roberta Cuel. The Role of Classification(s) in Distributed Knowledge Management. In *Proc. of KES*, Podere d'Ombriano, Crema, Italy, September 16-18 2002.
5. Jérôme Godard, Frédéric Andrès, William Grosky, and Kinji Ono. Knowledge Management Framework for the Collaborative Distribution of Information. In *Proc. of DataX (EDBT workshop)*, pages 2–16, Heraklion, Greece, March 14 2004.
6. Jérôme Godard, Frédéric Andrès, and Kinji Ono. Management of Cultural Information: Indexing Strategies for Context-dependent Resources. In *Proc. of the Nara Symposium for Digital Silk Roads*, pages 369–374, Nara, Japan, December 10-12 2003.
7. Matthias Grimm, Mohammed-Reza Tazari, and Dirk Balfanz. Towards a Framework for Mobile Knowledge Management. In *Proc. of PAKM*, pages 326–338, Vienna, Austria, December 2-3 2002.
8. Panu Korpipää and Jani Mäntyjärvi. An Ontology for Mobile Device Sensor-Based Context Awareness. In *Proc. of CONTEXT*, pages 451–458, Stanford, CA, USA, June 23-25 2003.
9. Kurt Maly, Mohammad Zubair, and Xiaoming Liu. Kepler - An OAI Data/Service Provider for the Individual. *D-Lib Magazine*, 7(4), April 2001.
10. Aloke Mukherjee, Babak Esfandiari, and Neal Arthorne. U-P2P: A Peer-to-Peer System for Description and Discovery of Resource-Sharing Communities. In *Proc. of ICDCS Workshops*, pages 701–705, Vienna, Austria, July 2-5 2002.
11. Kinji Ono, editor. *Proceedings of the Tokyo Symposium for Digital Silk Roads*, Tokyo, Japan, December 11-13 2001. UNESCO & National Institute of Informatics.
12. D.V. Sreenath, William Grosky, and Frédéric Andrès. *Intelligent Virtual Worlds: Technologies and Applications in Distributed Virtual Environments*, chapter Metadata-Mediated Browsing and Retrieval in a Cultural Heritage Image Collection. World Scientific Publishing Company, Singapore, 2002.
13. Eun Kyo Park Yugyung Lee, Changgyu Oh. Intelligent Knowledge Discovery in Peer-to-Peer File Sharing. In *Proc. of CIKM*, pages 308–315, McLean, Virginia, USA, November 4-9 2002.

Supporting Multi-Dimensional Trustworthiness for Grid Workflows

Elisa Bertino[1], Bruno Crispo[2], and Pietro Mazzoleni[3]

[1] Computer Sciences Dept. and
CERIAS Purdue University
bertino@cerias.purdue.edu
[2] Computer Science Department
Vrije Universiteit
crispo@cs.vu.nl
[3] Dipartimento di Informatica e Comunicazione
University of Milan, Italy
mazzolen@dico.unimi.it

Abstract. In this paper we present the problem of trustworthy computations in Grid systems. Trustworthy computations have several articulated requirements that are inroduced in the paper. As initial result, we introduce the notion that applications should be able to specify the criteria to be privileged when executing computation on a grid. This specification can be best specified for computations organized according to workflows in that it is possible to specify different criteria to be optimized for different tasks. In the paper we thus provide an initial definition of how a workflow specification can extended with the specification of the trustworthiness criteria to be optimized.

1 Introduction

Grid systems were initially developed for supporting scientific computations, in areas such as biotechnology, astronomy and physics, and therefore their purpose was mainly to support computationally intensive tasks. Today, companies, users and researchers are looking at ways to use the grid approach for commercial uses and for applications in many different areas, ranging from the entertainment to the financial industry. The development of techniques and tools for coordinating the execution of complex tasks over grids, such as workflow management systems, as well as the development of specialized web services and of storage services, is simplifying the deployment of grid-based applications.

However, a limiting factor to the wide exploitation of the computational grid paradigm is represented by the untrusted environment in which computations are sometimes to be performed. Not only a grid system may encompass a variety of nodes very much heterogeneous with respect to trust- some nodes may be trustworthy whereas others are not - but also the notion itself of trustworthiness is quite articulated. Trustworthiness may encompass several notions, like integrity, confidentiality, availability, privacy, reliability, to name some, and may

be evaluated according to different criteria and mechanisms. Since we cannot expect that a single notion be suitable for all the possible applications and users in a grid environment, we need a more flexible framework to assess and enforce trustworthiness which should allow users and applications to choose and select among various security requirements, for example to prioritize availability rather than confidentiality. It is also important to note that when a grid application is structured as a workflow process [1, 2], each of the composing steps may have different requirements with respect to trust.

In this paper, we elaborate on some of the concepts underlying such framework and point out relevant open research directions. This paper is not intended to be a complete solution to the problem, it is rather a first attempt to create a general framework to be used to develop trustworthiness-aware grid framework for uses in application spanning from biotechnology to multimedia data analysis and dissemination. The paper is organized as follows: Section 2 introduces the possible types of trustworthiness in Grid with special attention to Grid Workflow applications. Section 3 describes some research issues that have to be addressed in order to built a trust-aware grid workflow framework. Section 4 introduces the notion of Multi-Dimensional Trustworthiness for Grid Workflow, that is the basic element of our solution. Section 5 extends an existing workflow model with our notion of trustworthiness while Section 6 presents other grid workflow systems and compare them with our assumptions. The paper is concluded by Section 7 in which a summary of the solution as well as the future work is presented.

2 Grid trustworthiness

A unique characteristic of Grid provides an abstraction for resource sharing and collaboration across multiple administrative domains. However, resources in this environment span across multiple geographical locations, usually heterogeneous and administered by different resource owners. In this context, the notion of Trustworthiness results to be variegate and to include different aspects which are briefly described in what follows.

A first useful distinction concerns the resource to which the trust applies: we distinguish between data trustworthiness and computational trustworthiness.

Data trustworthiness refers to the trustworthiness of information stored into the Grid or generated by a computation. For instance, a user might want his/her data to be stored into a host which is always available, even if the node cannot perform fine grained access control to authorize/deny requests to the user data.

Computation trustworthiness refers to the trustworthiness of tasks to be executed by the Grid nodes. For example, a user might wish his/her tasks to be executed only at nodes which guarantee a high level of confidentiality or in which the availability of the service for the entire computation is guaranteed.

Both computation and Data trustworthiness can be furthermore classified based on the subject to which trust applies. We have resource owner trustworthiness and data&task owner trustworthiness.

Resource owner trustworthiness represents the trustworthiness of a node sharing physical resources (e.g., Data storage or computational power) across the Grid. It represents the requirements a host should satisfy in order to store data or to execute a task of a given user. For example, users might not want their data being stored by a node which is not reliable or their tasks computed by a host which does not adopt a specific security mechanism.

On the opposite, *data&task owner trustworthiness* represents the trustworthiness of a user executing tasks as well as storing or retrieving data into the grid. It represents the requirements a user should have in order to have his tasks executed (or data stored) by a given node. As an example, a node might not want data from untrusted (or possible malicious) users being stored into its resources.

Table 1 summarizes such four types of trustworthiness to be considerate with Grid. In this work, we consider those types of trust when the process to be executed over the Grid can be represented as a workflow.

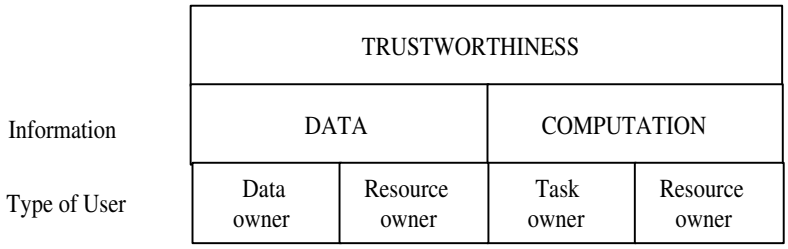


Fig. 1. Possible cases of trustworthiness in Grid

2.1 Grid Workflow

When a grid computation is organized according to a workflow, composed by several activities to be executed sequentially or in parallel by autonomous hosts, it is likely that security requirements of these activities about nodes storing data and nodes executing the computations using such data do not match. Thus, each task needs to identify suitable nodes in which to execute the computations. Nodes should be able to collect input data and to store output data from the Grid according to the various classes of trustworthiness described above. Moreover, to complete a workflow, information needs to migrate or being replicated into different hosts each satisfying a different subset of the trust requirements.

In such context, it is therefore important to have a comprehensive model dealing with all different types of trustworthiness at once. Figure 1 shows the main phases of executing a grid workflow. For sake of simplicity, we assume here a centralized system aware of all grid nodes available as well as the schema of the workflow to be executed. The first phase (step 1 in Figure 2) identifies the set of nodes in which to execute a task. Nodes are normally selected based on

user requests (amount of resource available, type of machine). In this phase, the computation trustworthiness is considered in order to filter the candidate nodes from the ones which do not fulfill trust requirements. Once a compliant node is identified, it loads data input eventually from another nodes into the Grid. Here data trustworthiness is considered (step 1 in Figure 2) to identify nodes collecting data having a trust level compliant to the one set by the user. The task can now be computed. Once the task is terminated, output data are stored back to a grid node again considering data trustworthiness (step 1 in Figure 2) and the central system can re-start the process with the following workflow task.

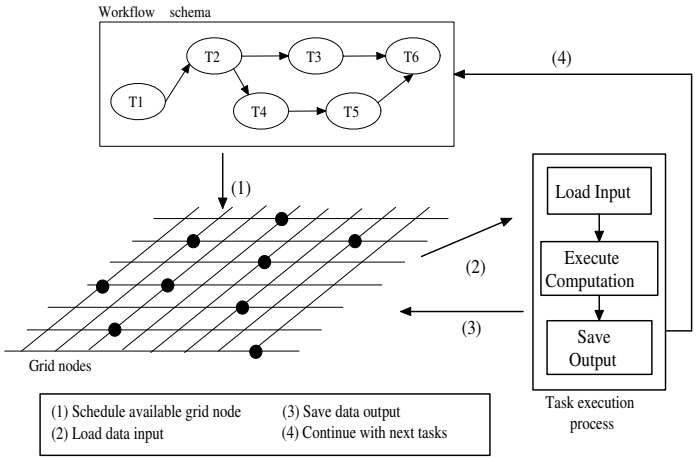


Fig. 2. Grid Workflow execution phase

3 Research Issues

In this paper, we elaborate on some of the concepts underlying such framework and point out relevant open research directions. In particular, our research is based on the following key concepts:

1. *An extensible set of grid node security properties* (integrity, confidentiality, availability, etc) - we assume that there is no specific single criteria. Rather there could be several security properties which are fixed by the user/applications and/or by the administrators of the grid. Security properties may be dynamically added and removed. Associated with each security property there will be a rating/label mechanism organized according to a partial order, that is, a lattice.

2. *Flexible evaluation methodology* - we do not assume that each grid node is rated according to all possible security properties. Rather we assume that each grid node may be rated according to an arbitrary subset of these properties or, even, not rated at all. Also, in our approach we assume that each rating criteria has associated one or more services allowing one to rate a given grid node with respect to the given security property. We assume both self-rating, according to which a node voluntarily rates itself for to a chosen set of security properties, and third-party rating, according to which the rating is performed by entities different from the rated node. The user is completely free to select which security property to evaluate and which evaluation services rely at any time accordingly to his/her trust requirements. He can even implements his own evaluation service.

3. *Trust-annotated workflow specification* - in order to support the specification of articulated trust requirements for grid , we assume that for each task of the workflow, the user may specify required conditions for one or more security property and also specify the approach used to evaluated those (i.e., subjective rather than using third part evaluation services, etc.).

4. *A scheduling algorithm with a set of relaxation strategies* - the scheduling algorithm is in charge of devising the set of nodes where the various workflow tasks are executed taking into consideration, among the others, the trust-annotated specification. Since depending on the status of the grid and on the specification, it may not be always possible to find an execution schedule, strategies should be provided in order to relax some of the constraints and/or to modify the workflow. In our solution, constraints can be relaxed or even better negotiated between parties.

4 Context Lattices

As discussed in the introduction, trustworthiness in Grid may encompass several aspects, like integrity, confidentiality, authorization, availability, reliability, privacy, and so on. Therefore, the first issue to be investigated is the development of a structure to enforce multi-dimensional quality of services for grid workflows. In the following, we refer to such structure as a *Context Lattice*. A context lattice is a set of quality of service criteria (or dimensions) which can be associated with an host participating to the Grid. A context lattice is a combination of values for one or more lattice dimensions.

A lattice dimension l_i is defined as a tuple $\langle name_i, Value_i, \prec_i \rangle$ where $name_i$ defines a unique identifier, $Value_i$ is a set $\{v_1, v_2, \ldots, v_n\}$, and $\prec_i$ defines a local order among values. We say that v_i precedes v_j in the order if $v_i \prec v_j$.

Examples of lattice dimensions are the following:
$\langle trust, \{a..z\}, alphabetical \rangle\}$, $\langle privacy, \{high, medium, low\}, \{high \prec medium \prec low\} \rangle$.

By using lattice, each host is therefore classified according to multiple-dimensions which are used for selecting nodes in which to execute some activities or to store

some output. For example, given the two dimensions presented above, a context lattice for a task t_i could be $\langle [privacy, high], [trust, a] \rangle$ which specifies t_i should run into a node having a high level of trust as well as a very high level of privacy. In this context, we assume a lattice being specified for each node (host or user) of the Grid. Lattice values for a task can be assigned using different strategies such as directly assigned by a trusted party, automatically inferred by the history or generated based on a combination of reputation and trust [3].

Moreover, given n possible dimensions, a task can be associated with n lattices values (i.e. one for each dimension) as well as a subset of n. In case a dimension is not available, we assume to automatically set its value to the lowest possible value (such as privacy=low and trust=z in our example).

5 Workflow Model

To apply our solution, we adopt the existing workflow model proposed in [4] because it is formal and flexible enough to describe our approach. However, our solution can be applied to any other workflow models.

As in most WFMSs, a workflow is defined as a set of tasks with tasks dependencies defined among them. Formally, a workflow W is defined as a pair $\langle T, D \rangle$ where T denotes set of tasks $t_1, t_2, \ldots, t_n$ composing the workflow and D denotes the set of intertask dependencies among tasks.

A workflow task, or activity, describes a piece of work that forms one logical step within a process which can be executed manually or automatically [**?**]. The host executing the task accepts as input a set of information, processes them and produces some results as output. Input data can be collected from the user, loaded from the outputs of previous tasks or by using information already available within the Grid. Similarly, output data can be sent to the user, saved temporarily for being used by following tasks in the same workflow or saved permanently into the Grid and shared among other users. A task can be formally defined by a tuple $\langle Act, Host, I, O \rangle$ in which Act represents the list of activities to be executed, $Host$ the node in which the task has to be processed, and I and O represent Input and Output data respectively. In a Grid environment, it is reasonable to assume many hosts available to execute a certain task. The user does not specify in advance the host in which to execute a task and *host* is replaced with the information about the set of resources (software and hardware) needed for the task to be properly executed. The node (or set of nodes) where the task is physically executed will be selected at runtime by the resource allocation process.

A workflow intertask dependency describes the precedence order among tasks and the conditions when a task can be executed. A dependency has the form $t_i \xrightarrow{x} t_j$ states that the task t_j can start after (or along with) task t_i when the condition x is verified. x represents the dependency type and it is defined as a logical expression which specify the conditions under which a task can be executed. An extensive list of the variety of dependencies supported by a workflow

can be find in [4]. In the following, we do not further explore dependencies and in case of $t_i \xrightarrow{x} t_j$ we assume t_j can be executed when t_i is successful terminated.

In order to support multi-dimensional level of trust, we assume that for each workflow task the user specifies the required values for one or more rating criteria both for the computational and data trustworthiness.

Therefore, task definition needs to be extended to include contexts. Formally, a workflow task t_i is defined as a tuple $\langle Act, \{Host, cont_{proc}\}, \{I, cont_{dataI}\}, \{O, cont_{proc}\}\rangle$ in which Act, $host$, I and O are the same as previous whereas $cont_{proc}$, represents computational quality, and $cont_{dataI}$ and $cont_{dataO}$ the Input and Output data quality respectively.

In a same way, we assume each grid host being able to specify trust requirements for users who are willing both to store data and execute computation using his/her shared resources.

Such information is the foundations onto which we apply the multi-dimensional trustworthiness criteria. Through our notion of lattices, that can be applied to both the *host* (for Computational trustworthiness) and the I and O (for Data trustworthiness) of each task of the workflow, users can select multiple trust criteria. A host will be selected by the scheduler for executing a task (as well as for storing information) only if it satisfies the security conditions set by the user on behalf of whom the workflow is executed. However, we also need to deal with cases in which such conditions cannot be satisfied. As an example, consider a scenario in which a user specifies that the input data for a task t_i should be loaded from nodes whose lattices has a level of confidentiality sets at least as "high". If the input data is available only from a node with confidentiality="medium", an inconsistency arises. To solve such inconsistency, an immediate solution could be migrating input to another node matching the condition. However, the new node storing data should match all the security conditions by the task that generate the data (e.g., store data only into nodes which availability="high"). In case of several tasks sharing the same data, there might problems in planning a suitable assignment without storing information only onto grid nodes having lattices with higher values for all the possible criteria. In our work, we do not have trust as stand-alone component for grid workflows, instead we develop a solution which takes into consideration other elements such as keeping minimal the number of data-migrations and balancing workload distribution among grid nodes having different security properties.

6 Related Work

During the last few years, Workflow is gaining always more and more attention within the Grid community. In fact, the original idea of Grid to have a flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions, and resources as in [5], shows expectations which go beyond the "simple" execution of a time consuming computation or the storage of large amount of data into a highly distributed environment. There is the need of using Grid for more complex processes, in which information or data are passed

from one grid resource to another for further analysis, according to a set of procedural rules. The current efforts direct to finalize and adopt the Open Grid Service Architecture (OGSA) [6], the Grid architecture based on Web service standards, shows how Workflow technology and Grid are merging. Even if most of the problems are still open, some work has been appeared to leverage OGSA with the concepts of workflow management systems, with particular attention to the Business Process Execution Language for Web Services (BPEL4WS).

One of the most well known projects that addresses workflow management in an OGSA environment is the Grid Services Flow Language [2], GSFL which proposes an XML based language supporting workflow specifications and an execution engine. However, the project is based on a previous version of Globus Toolkit, GT, and does not consider any WS-Resource Framework introduced into the latest releases of GT which can be considerate the de-facto standard.

Another interesting solution is the Grid Workflow Execution Language, GWEL [7]. GWEL is a XML based language built for integrating grid workflow with any GT3 compatible service. The proposed solution is based on the fact that hosts offering services as well as hosts collecting data are made available to users willing to create workflows into two sets called Factory Links and Data Links respectively. Workflow schemas created combining those two sets with other elements (such as control flow or lifecycle elements) are then given as input to the engine which automatically executed it over the Grid.

Similar goals are pursued by another system, GridAnt [8] which explores the needs of a Grid user to map his processes to the Grid nodes. The solution, based on Java, makes use of a Workflow dictionary to define the operations (grid-execute, grid-copy, grid-query) to be executed over a set of predefined tasks mapped on available grid nodes.

Even if the above solutions address some grid-workflow problems, they go in a different direction with respect to our notion of Grid workflow according to which user does not know in advance the exact hosts in which he wants tasks to be executed or his data to be stored or loaded for computation. User simply specifies, for each task, his requirements (e.g, amount and type of resource needed to execute a task) along with the the trust criteria he wants to adopt. The system automatically collects such information to the compliant hosts which can execute a task among the ones available.

7 Conclusions and Future work

In this paper we have presented an ongoing work developing a Multi-Dimensional Trustworthiness for Grid Workflow. The problem is presented along with some of main research directions we have taken. The work starts from the need of having a Grid workflow whose tasks should not be executed over predefined Grid nodes rather than the ones available which offers a better level of trustworthiness. Trustworthiness requirements which could be different based on the workflow tasks or the criteria are used to compute the value of each host. A classification

of trustworthiness types are described, and a first attempt to formalize the notion of multi-level trustworthiness, such as the lattices, is presented.

As future work, we plan to implement the proposed framework in the context of current workflow and web service standards. We also plan to investigate how our framework can be used for providing trusted data and query management services as well as ensuring user privacy for digital libraries exploiting grid infrastructures.

8 Acknowledgement

This work has been partially supported by DELOS, a Network of Excellence in Digital Libraries.

References

1. Junwei Cao and Stephen A. Jarvis and Subhash Saini and Graham R. Nudd. GridFlow: Workflow Management for Grid Computing. 3rd International Symposium on Cluster Computing and the Grid, 2003
2. Sriram Krishnan and Patrick Wagstrom and Gregor von Laszewski. GSFL: A Workflow Framework for Grid Services", Argonne National Laboratory, Preprint ANL/MCS-P980-0802, Aug 2002
3. F. Azzedin and M. Maheswaran.Integrating Trust into Grid Resource Management Systems.In International Conference on Parallel Processing (ICPP'02), Vancouver, B.C., Canada,2002"
4. E. Bertino and Elena Ferrari and V. Atluri. The Specification and Enforcement of Authorization Constraints in Workflow Management Systems. TISSEC 2(1): 65-104, 1999
5. Ian Foster and Carl Kesselman and Steven Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Lecture Notes in Computer Science, 2001
6. Foster, I., Kesselman, C., Nick, J. M., and Turcke, S., The physiology of the Grid: An Open Grid services Architecture for distributed systems integration. Open Grid Service Infrastructure WG, Global Grid Forum, 2002
7. Dieter Cybok.A Grid Workflow Infrastructure. GGF10 Workshop, Berlin May 2004
8. Kaizar Amin, Gregor von Laszewski, Mihael Hategan, Nestor J. Zaluzec, Shawn Hampton, Alberto Rossi.GridAnt: A Client-Controllable Grid Work.ow System. Proceedings of the 37th Hawaii International Conference on System Sciences,2004.

Query Trading in Digital Libraries

Fragkiskos Pentaris and Yannis Ioannidis

Department of Informatics and Telecommunications, University of Athens,
Panepistemioupolis, 15771, Athens, Greece
E-mail:{frank,yannis}@di.uoa.gr

Abstract. In this paper we present a distributed query framework suitable for use in federations of digital libraries (DL). Inspired by e-commerce technology, we recognize queries (and query answers) as commodities and model the task of query processing as a task of trading queries and queries-answers. We show that our framework satisfies the needs of modern DL federations by respecting the autonomy of DL nodes and natively supporting their business model. Our query processing conception is independent of the possible distributed architecture and can be easily implemented over a typical GRID architectural infrastructure or a Peer-To-Peer network.

1 Introduction

Digital Libraries' users may need to simultaneously use two or more libraries to find the information they are looking for. This increases the burden of their work as it forces them to pose the same query to different DLs multiple times, each time using a possibly different user interface and a different metadata schema. Digital Libraries are aware of this deficiency and have been trying for a long time to attack this problem by creating federations of DLs. Especially for small DLs, joining such federations is necessary for attracting more users and thus, ensuring their economic survival.

The architectures of these federations usually follow a wrapper-based centralized mediation approach. Nevertheless, current growth of DL collections and increase in the number of DLs joining a single federation have rendered these architectures obsolete. Almost every major DL is evaluating new architectures to replace its old systems. For instance, a kind of P2P architecture will be evaluated within the framework of the BRICKS European Integrated Project (peer nodes are called Bricks nodes in this project), whereas the GRID architecture will be evaluated within the DILIGENT project (The University of Athens participates in both consortiums).

Obviously, existing DL federations will benefit a lot by the above architectures as the improved search and browse response-time will enable them to form even larger federations. On the other hand, even today's hardware and software architectures (e.g., ultra fast SANs and SMP machines) do provide the means for building fast federations, yet DLs are still reluctant in unconditionally joining them. It seems that apart from the scalability problem, there are additional ones that inhibit the creation of large federations. DLs prefer to keep their systems completely autonomous. They want their nodes to be treated as black boxes, meaning that remote nodes should make no assumption on the contents, status and capabilities of their systems. Exporting this information to

distant nodes hurts the autonomy of DLs, which in turn reduces their flexibility and increases the burden of their work. An additional problem is that DLs are usually competitive institutions, therefore, the proposed distributed architectures should natively respect and support their business requirements.

The main contribution of this position paper is the presentation of a query processing schema, which may be setup over a P2P or GRID-based network architecture. Our proposal respects the autonomy of existing DLs and natively addresses their business model. The rest of the paper is organized as follows: In section 2, we discuss the business requirements of DLs. In section 3, we present our query processing architecture.

2 Digital Libraries Federations Requirements

In the introduction we argued that DLs are reluctant in forming federations because they are not sure that these systems comply with their business model and respect their autonomy. In the following paragraphs, we briefly examine these requirements focusing on the problems they create to the two most prominent future DL architectures, i.e, P2P networks and the GRID.

Business Model - Content Sharing Economic prosperity of Digital Libraries is usually bound to their ability to sell information (content, annotation and metadata) and data processing services to their users. These are in fact the only assets that DLs hold, making them reluctant to give away any data to third-party entities, especially if this is done over the Internet. For instance, in BRICKS many institutions will not export or mirror their data to the common BRICKS community network but instead will allow access to their data and legacy systems through the use of conventional wrappers. Employing a strict security and trust policy in every network node and using state-of-the-art content-watermarking techniques reduces the objections of DL in sharing their data. Nevertheless, experience shows that no security system is perfect and DLs are aware of this fact.

The reluctance of DL to share their content creates a lot of problems in architectures following the GRID paradigm, since the latter model the process of evaluating queries as a task of moving the data (content) to one or more processing GRID-nodes, and then executing the distributed query execution plans. Obviously, a completely new query processing architecture must be used that will minimize the physical movement of (unprocessed) data.

P2P-based systems are also affected by the content sharing restriction problem. Building a metadata P2P indexing service, using a Distributed Hash Table (DHT), will distribute the metadata of a DL to multiple, potentially less trusted nodes, including other competitive DLs. This is not something that a DL's manager will easily approve. A solution would be to use a double hashing technique, i.e., build a DHT of the hash value of the metadata instead of the metadata themselves. In this way, nodes will know the hash value of the metadata but not the exact metadata. If the users make only keywords-based queries, this approach will be satisfactory. But if advanced query capabilities are required, a traditional query processor that uses DHT indices and requires the physical movement of data, just like the GRID architecture, will have to be used. Thus, even in

the case of P2P-based systems, a new query processing framework is required that will also minimize the physical movement of unprocessed information.

Business Model - Integration of Query Processing and Accounting Consider a small federation of two DLs where the first DL holds digital pictures while the second one has information concerning poetry. Assume that a user of this federated system is looking for pictures that were created by painters that have also written certain types of poems. This query combines pieces of information from both DLs, yet only retrieves/browses content from the first one. Since DL sell (any possible piece of) information, it is a matter of the billing policy of each DL, whether the user should be charged only for the retrieved content, or also for the information of the second DL that was used during query processing. If DLs choose to charge for any information they provide, which we expect in the future to be a typical business scenario, the query optimizer and the accounting system should be closely integrated.

Competitive Environment The most important business requirement of future DLs federations is that these should be compatible with the competitive nature of the DLs market, i.e., information is asset and data should be treated as commodities for trading in a competitive environment. Competition creates problems in DL federations, as it results in potentially inconsistent behavior of the nodes at different times. The query processing architecture should be capable of handling cases where remote nodes expose imprecise information. Such behavior is typical (and allowed) in competitive environments.

Autonomy A requirement of modern DL federations is that distributed architectures should respect the autonomy of DLs and treat them as black boxes. Middle-wares and wrapper-based architectures help in preserving the autonomy of remote nodes. However, during query processing and optimization, existing proposals require, *a priori*, certain pieces of information (e.g., data statistics, remote nodes status (workload), nodes capabilities (e.g., which variables must be bound), operators (e.g., union, join) cost functions and parameters) that clearly violate their autonomy. A proper query processing architecture that respects DL's autonomy and work only with information that nodes expose during query processing.

3 The Query Trading architecture

3.1 Overview

We have recently proposed a new query processing architecture [9] that meets the scalability and autonomy requirements of future DLs and perfectly matches their business requirements. We treat DL nodes as black boxes, assuming nothing on the contents and capabilities of each node. Execution of distributed queries is handled by splitting them into pieces (sub-queries), forwarding them to nodes capable of answering them, and then combining the results of these queries to build the answer of the distributed query.

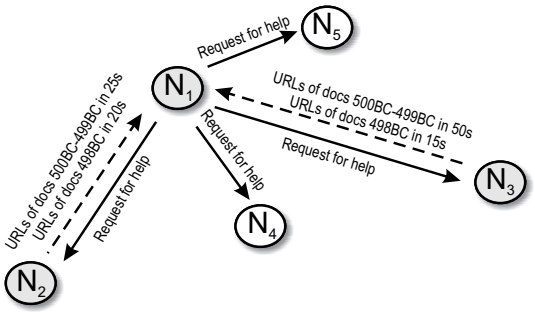


Fig. 1. Example of query processing.

To make our architecture more concrete, consider the case of a large federation of libraries (Fig. 1) and assume that a user at node N_1 asks for the URLs of every ancient Greek document that was written in Athens between 500BC and 498BC. Since DL nodes are black boxes, N_1 can do nothing better than asking the rest DL nodes for any piece of information that might be of some help in evaluating the query. In this example, assume that two nodes, N_2 and N_3, have some relevant information and offer to help N_1. Node N_2 offers to return to N_1 a URL list of all relevant documents that were written between 500BC and 499BC in 25 seconds and the rest documents (498BC) in 20 seconds. Similarly, node N_3 offers the same information in 50 and 15 seconds respectively. Obviously, node N_1 query optimizer will choose node N_2 for all URLs concerning documents written in 500-498BC and N_3 for the rest ones. That is, the query processor of N_1 effectively purchases the answer of the original query from nodes N_2 and N_3 for 20 and 15 seconds respectively.

The above example shows the main idea behind the query processing architecture that we propose. It is inspired by e-commerce technology, recognizes queries (and query answers) as commodities and approaches DL federations as information markets where the commodities sold are data. Query parts (and their answers) are traded between DL nodes until deals are struck with some nodes for all of them. Distributed query execution is thus modeled as a trading of parts of query-answers. Buyer nodes (e.g., N_1) are those requiring certain pieces of information in order to answer a user query. Seller nodes (e.g., N_2 and N_3) are those offering buyers this missing information.

Although the idea is simple, it is difficult to construct an algorithm that can automate the trading of queries and queries answers. For instance, assume that in the previous example, node N_3 offered the URLs of the documents written in 500BC,499BC and 498BC separately for 20s, 30s, and 15s respectively. In this case, node N_1 has many different ways of combining the offers of N_2 an N_3. In fact, it might worth for N_1 to negotiate with node N_2 the case of N_2 also returning the URLs of the documents written in 500BC and 499BC separately, before node N_1 decides on which offers can be combined in the best way.

3.2 General Trading Negotiation Parameters

There are a lot of parameters that affect the performance of a trading framework such as the one described in the previous sub-section. For details on these parameters see [1–3, 5–9, 11, 12, 14]. We briefly describe the most important ones:

Negotiation protocol Trading negotiation procedures follow rules defined in a negotiation protocol [14]. In each step of the procedure, the protocol designates a number of possible actions (e.g., make a better offer, accept offer, reject offer, etc.) that a node may take. In the previous example, we assumed that the protocol used was *bidding* [13]. This protocol is simple but obviously cannot work when the number of nodes are too many. In larger networks a better alternative is to use an agent-based or P2P-based [7] *auction*. If the items/properties negotiated are minor and the nodes participating in the negotiation are few, then the oldest known protocol, *bargaining* can be also used.

Strategy In each step of the negotiation procedure and depending on the negotiation protocol followed, nodes have multiple possible actions to choose from. It is the *strategy* followed by each node that designates which action is the best one. The strategy can be either cooperative or competitive (non-cooperative). In the first case, nodes try to maximize the join utility of all nodes that participate in the negotiation. In the second case, nodes maximize their personal utility. Our architecture supports both types of strategies. In cooperative ones, nodes expose information that is accurate and complete. In competitive setups, nodes expose information that is usually imprecise. For instance, a node may lie about the time required for the retrieval of a piece of information.

User preferences and items valuation In section 3.1 we gave an example where the value of the commodities (i.e., the pieces of information) offered by remote nodes was expressed in term of the time required to fetch this information. More generally, offers of remote nodes will have many different properties, including (e.g.) the time required to retrieve the information, the precision and age of the data, and its cost in monetary units. That is, the valuation of an offer is multi-dimensional (a vector of values). The user must supply a preference relation over the domain of these vectors that orders the set of possible offers. This relation is known to buyer nodes and is used during the negotiation phase (e.g., during bidding) to select the offers that best fit the needs of the user.

Market Equilibrium In competitive environments, nodes provide imprecise information. For instance, if the preference relation is the total cost (in monetary units) of the answer, then nodes will increase the value of all pieces of information that have more demand than supply. This will cause a decrease in the demand of this information and after some time, values will stop fluctuating and the market will be in equilibrium (This requires all other parameters affecting the value of items to be static) [4]. The designer of a system can model the market in such a way that equilibrium values force the system to have a specific behavior (e.g, altruistic nodes are not overloaded). A nice property of our architecture is that according to the first welfare theorem [10], equilibriums will always be Pareto optimal, i.e., no node can increase its utility without decreasing the utility of at least on other node.

Message congestion mechanisms Distributed implementation of the previous negotiation protocols have run into message congestion problems [13] caused by offers flooding. This can be avoided using several different approaches such as agent-based architectures, focused addressing, audience restriction, use-based communication charges and, mutual monitoring [8, 13].

3.3 The Proposed Architecture

As it was mentioned earlier, we model query processing as a query trading procedure. Although there is a lot of existing work in e-commerce and e-trading (see previous subsection), there in an important difference between trading queries (and their answers) and the rest commodities. In traditional e-commerce solutions, the buyer receives offers for the complete items that he/she has asked for. However, in our case, it is possible that no DL node has every piece of information required for answering a user supplied query. Sellers will have to make offers for parts of the query (sub-queries) depending on the information that each DL holds locally. Buyers will have to somehow merge these offers to produce the answer of the initial queries. Since all nodes are black boxes, most sellers will make overlapping offers and buyers will have to make multiple rounds of communication with the seller nodes to ensure that the accepted offers are not overlapping. The problem of query optimization also complicates the task of the buyer since better offers not always improve the global distributed query execution plan. In the next paragraphs, we present how query optimization works in our framework. Further details on the proposed framework and its performance characteristics are given in [9].

The distributed execution plans produced by our framework consist of the query-answers offered by remote DL seller nodes together with the processing operations required to construct the results of the optimized queries from these offers. The query optimization algorithm [9] finds the combination of offers and local processing operations that minimizes the valuation (cost) of the final answer. For this reason, it runs iteratively, progressively selecting the best execution plan. In each iteration, the buyer node asks (Request for Bids -RFBs) for some queries and the sellers reply with offers that contain the estimations of the properties of these queries (query-answers). Since sellers may not have all the data referenced in a query, they are allowed to give offers for only the part of the data they actually have. At the end of each iteration, the buyer uses the received offers to find the best possible execution plan, and then, the algorithm starts again with a possibly new set of queries that might be used to construct an even better execution plan. The buyer may contact different selling nodes in each iteration, as the additional queries may be better offered by other nodes. This is in contrast to the traditional trading framework, where the participants in a negotiation remain constant.

In order to demonstrate our algorithm, we will use Fig. 2 that shows a typical message workflow among the buyer and seller nodes when the number of nodes is small (i.e., the bidding protocol is sufficient and we don't need to use auctions) and nodes follow a cooperative strategy. In this figure, a node receives a query Q that cannot be answered with the data that this node locally holds. For this reason it acts as a buyer node and broadcasts a RFB concerning query Q to some candidate seller nodes. The sellers in their turn, examine the query and if they locally have any relevant information

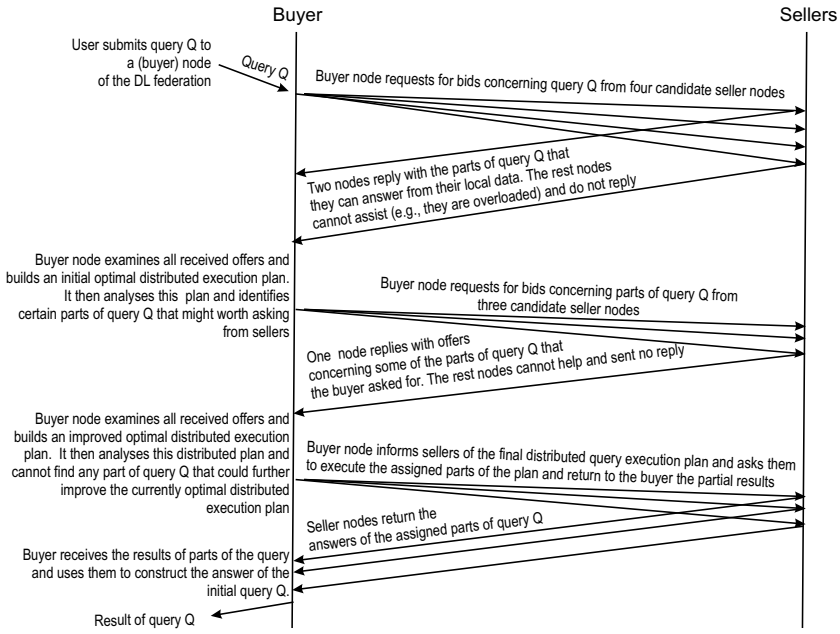


Fig. 2. Workflow of network messages between the buyer and seller nodes.

concerning parts of it, they inform the buyer of the properties of these parts (offers). In our example, only two nodes return some offers back to the buyer.

The buyer node waits for a timer to expire (bidding duration) and then considers all offers it has received to construct an initial optimal distributed query execution plan. It then examines this plan to find any other possible part of the query that could help the buyer further improve the distributed plan. In Fig. 2 we assume that the buyer (e.g.) found some parts of the initial query that are offered by both sellers. For this reason it starts a second iteration of the bidding procedure, this time requesting for bids on these overlapping parts. Figure 2 shows that only one seller makes an offer in the second bidding procedure. After the bidding procedure of the second negotiation is over, the buyer uses the new offer(s) to further improve the distributed plan and then re-examines it to find any other possible part of the query that can be improved. In our example, we assume that the buyer cannot find any such sub-query. Therefore, it asks from the selected remote nodes to evaluate the parts of the distributed plan that have been assigned to them and then return the results of these parts back to the buyer node. The latter uses these results to construct the answer of the initial query Q.

The above example is a rather simple case. In a more complex scenario, the nodes may be allowed to make offers which contain parts that have been sub-contracted to third nodes. The negotiation protocol may include bargaining and complex contracting details. Nodes may follow a competitive strategy where sellers/buyers may *strategically* delay or even refuse to make offers in an attempt to raise/drop the values of their offers. These complex scenarios are beyond the scope of this paper.

4 Conclusion

We propose a query processing paradigm, that respects the autonomy of DL nodes and natively supports their business model (information trading). Our query processing conception is independent of the possible distributed architecture. For instance, it can be easily implemented over a typical GRID architectural infrastructure, where the GRID nodes will act as sellers and/or buyers of information. A different option is to use a decentralized (P2P) agent-based auction mechanism. Our framework can also extend pure P2P-based architectures to support advanced queries instead of plain keywords-based queries.

References

1. M. Bichler, M. Kaukal, and A. Segev. Multi-attribute auctions for electronic procurement. In *Proc. of the 1st IBM IAC Workshop on Internet Based Negotiation Technologies, Yorktown Heights, NY, March 18-19*, 1999.
2. John Collins, Maksim Tsvetovat, Rashmi Sundareswara, Joshua van Tonder, Maria L. Gini, and Bamshad Mobasher. Evaluating risk: Flexibility and feasibility in multi-agent contracting. In *Proc. of the 3rd Annual Conf. on Autonomous Agents , Seattle, WA, USA.*, May 1999.
3. V. Conitzer and T. Sandholm. Complexity results about nash equilibria. *Technical report CMU-CS-02-135*, `http://www-2.cs.cmu.edu/~sandholm/Nash_complexity.pdf`, 2002.
4. Donald Ferguson, Christos Nicolaou, and Yechiam Yemini. An economy for managing replicated data in autonomous decentralized systems. In *Proc. of Int. Symposium on Autonomous and Decentralized Systems*, 1993.
5. John H. Kagel. *Auctions: A Survey of Experimental Research.* The Handbook of Experimental Economics, edited by John E. Kagel and Alvin E. Roth, Princeton: Princeton University Press, 1995.
6. Sarit Kraus. *Strategic Negotiation in Multiagent Environments (Intelligent Robotics and Autonomous Agents).* The MIT Press, 2001.
7. E. Ogston and Stamatis Vassiliadis. A Peer-to-Peer Agent Auction. In *Proc. of AAMAS02, Bologna, Italy*, July 15–19 2002.
8. H. Van Dyke Parunak. *Manufacturing experience with the contract net.* Distributed Artificial Intelligence, Michael N. Huhns (editor), Research Notes in Artificial Intelligence, chapter 10, pages 285-310. Pitman, 1987.
9. Fragkiskos Pentaris and Yannis Ioannidis. Distributed query optimization by query trading. In *Proc. of Int. Conf. on Extending Database Technology (EDBT), Herakleio, Greece*, 2003.
10. Mark Pingle and Leigh Tesfatsion. Overlapping generations, intermediation, and the first welfare theorem. *Journal of Economic Behavior and Organization.*, 3(5):325–345, 1991.
11. J. S. Rosenchein and G. Zlotkin. *Rules of Encounter : designing conventions for automated negotiation among computers.* The MIT Press series in artificial intelligence, 1994.
12. Tuomas Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artificial Intelligence*, 135:1–54, 2002.
13. Reid G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, 29(12):1104–1113, December 1980.
14. Stanley Y.W. Su, Chunbo Huang, Joachim Hammer, Yihua Huang, Haifei Li, Liu Wang, Youzhong Liu, Charnyote Pluempitiwiriyawej, Minsoo Lee, and Herman Lam. An internet-based negotiation server for e-commerce. *VLDB Journal*, 10:72–90, 2001.

Scalable Similarity Search in Metric Spaces

Michal Batko[1], Claudio Gennaro[2], Pasquale Savino[2], and Pavel Zezula[1]

[1] Masaryk University, Brno, Czech Republic
{zezula, xbatko}@fi.muni.cz
[2] ISTI-CNR, Pisa, Italy
{claudio.gennaro, pasquale.savino}@isti.cnr.it

Abstract. Similarity search in metric spaces represents an important paradigm for content-based retrieval of many applications. Existing centralized search structures can speed-up retrieval, but they do not scale up to large volume of data because the response time is linearly increasing with the size of the searched file. The proposed GHT* index is a scalable and distributed structure. By exploiting parallelism in a dynamic network of computers, the GHT* achieves practically constant search time for similarity range queries in data-sets of arbitrary size. The amount of replicated routing information on each server increases logarithmically. At the same time, the potential for interquery parallelism is increasing with the growing data-sets because the relative number of servers utilized by individual queries is decreasing. All these properties are verified by experiments on a prototype system using real-life data-sets.

1 Introduction

The search operation has traditionally been applied to structured (attribute-type) data. Complex data types – such as images, videos, time series, text documents, DNA sequences, etc. – are becoming increasingly important in modern digital libraries. Searching in such data requires a gradual rather than the exact relevance, so it is called the *similarity* retrieval. Given a query object q, this process involves finding objects in the database D that are similar to q. It has become customary to assume the similarity measure as a distance metric d. The primary challenge in performing similarity search is to structure the database D in such a way so that the search can be performed fast.

Though many metric index structures have been proposed, see the recent surveys [2] and [6], most of them are only main memory structures and thus not suitable for a large volume of data. The scalability of two disk oriented metric indexes (the M-tree [3] and the D-index [5]) have recently been studied in [4]. The results demonstrate significant speed-up (both in terms of distance computations and disk-page reads) in comparison with the sequential search. Unfortunately, the search costs are also linearly increasing with the size of the data-set.

On the other hand, it is estimated that 93% of data now produced is in a digital form. The amount of data added each year exceeds exabyte (i.e. 10^{18}

bytes) and it is estimated to grow exponentially. In order to manage similarity search in multimedia data types such as plain text, music, images, and video, this trend calls for putting equally scalable infrastructures in motion. In this respect, the Grid infrastructures and the Peer-to-Peer (P2P) communication paradigm are quickly gaining in popularity due to their scalability and self-organizing nature, forming bases for building large-scale similarity search indexes at low costs. However, most of the numerous P2P search techniques proposed in the recent years have focused on the single-key retrieval. See for example the *Content Addressable Network* (CAN) [9], which is a distributed hash table abstraction over the Cartesian space.

Our objective is to develop a distributed storage structure for similarity search in metric spaces that would scale up with (nearly) constant search time. In this respect, our proposal can be seen as a *Scalable and Distributed Data Structure* (SDDS), which uses the P2P paradigm for the communication in a Grid-like computing infrastructure. We achieve the desired effects in a given (arbitrary) metric by linearly increasing the number of network nodes (whole computers), where each of them can act as a client and some of them can also be servers. Clients insert metric objects and issue queries, but there is no specific (centralized) node to be accessed for all (insertion or search) operations. At the same time, insertion of an object, even the one causing a node split, does not require immediate update propagation to all network nodes. A certain data replication is tolerable. Each server provides some storage space for objects and servers also have the capacity to compute distances between pairs of objects. A server can send objects to other peer servers and can also allocate a new server.

The rest of the paper is organized as follows. In Section 2, we summarize the necessary background information. Section 3 presents the GHT* distributed structure and its functionality. Section 4 reports some results of our performance evaluation experiments. Section 5 concludes the paper and outlines directions for future work.

2 Preliminaries

Probably the most famous scalable and distributed data structure is the LH* [8], which is an extension of the *linear hashing* (LH) for a dynamic network of computers enabling the *exact-match* queries. The paper also clearly defines the desirable properties of SDDSs in terms of *scalability*, *no hot-spots*, and *update independence*.

2.1 Metric Space Searching Methods

The effectiveness of metric search structures [2,6] consists in their considerable *extensibility*, i.e. the degree of ability to support execution of a variety of queries. Metric structures can support not only the exact-match and range queries on sortable domains, but they are also able to perform similarity queries in the

generic metric space, considering the important *Euclidean* vector spaces as a special case.

The mathematical *metric space* is a pair (D, d), where D is the *domain* of objects and d is the *distance function* able to compute distances between any pair of objects from D. It is assumed that the smaller the distance, the closer or more *similar* the objects are. For any distinct objects $x, y, z \in D$, the distance must satisfy the following properties of *reflexivity*, $d(x, x) = 0$, *strict positiveness*, $d(x, y) > 0$, *symmetry*, $d(x, y) = d(y, x)$, and *triangle inequality*, $d(x, y) \leq d(x, z) + d(z, y)$.

Though several sophisticated metric search structures have been proposed in literature, the two fundamental principles are called the *ball* and the *generalized hyperplane* partitioning [10]. In this article, we have concentrated on the recursive application of the second principle, which can be seen as the *Generalized Hyperplane Tree* (GHT). For the sake of clarity of the further discussion, we describe the main features of the GHT in the following.

Generalized Hyperplane Tree – GHT The GHT is a binary tree with metric objects kept in leaf nodes (buckets) of fixed capacity. The internal nodes contain two pointers to descendant nodes (sub-trees) represented by a pair of objects called the *pivots*. In the leaf nodes of the left sub-tree are objects closer to the first pivot, objects closer to the second pivot are in the leaf nodes of the right sub-tree.

The creation of the GHT structure starts with the bucket B_0. When the bucket B_0 is full, we create a new empty bucket, say B_1, and move some objects from B_0 to B_1. This idea of split is implemented by choosing a pair of pivots P_1 and P_2 ($P_1 \neq P_2$) from B_0 and by moving all the objects O, which are closer to P_2 than to P_1, into the bucket B_1. The pivots P_1 and P_2 are then placed into a new root node and the tree grows by one more level. In general, given an internal node i of the GHT structure with the pivots $P_1(i)$ and $P_2(i)$, the objects that meet Condition (1) are stored in the right sub-tree. Otherwise, they are found in the left sub-tree.

$$d(P_1(i), O) > d(P_2(i), O). \tag{1}$$

To **Insert** a new object O, we first traverse the GHT to find the correct storage bucket. In each inner node i, we test Condition (1): if it is true, we follow the right branch. Otherwise, we follow the left one. This is repeated until a leaf node is found and the object O is inserted – the split is applied if necessary.

In order to perform a similarity **Range Search** for the query object Q and the search radius r, we recursively traverse the GHT following the left child of each inner node if $d(P_1(i), Q) - r \leqslant d(P_2(i), Q) + r$ is satisfied and the right child if $d(P_1(i), Q) + r > d(P_2(i), Q) - r$ is true. Observe that, depending on the size of the radius r, both the conditions can be met simultaneously, which implies necessity of searching the left and the right subtrees at the same time.

3 GHT*

In general, the scalable and distributed data structure GHT* consists of a network of nodes that can insert, store, and retrieve objects using similarity queries. The nodes with all these functions are called *servers* and the nodes with only the insertion and query formulation functions are called *clients*. The GHT* architecture assumes that network nodes communicate through the *message passing* paradigm. For consistency reasons, each *request message* expects a confirmation by a proper *reply message*. Each node of the network has a unique *Network Node IDentifier* (NNID). Each server maintains data objects in a set of *buckets*. Within a server, the *Bucket IDentifier* (BID) is used to address a bucket. Each object is stored exactly in one bucket.

An essential part of the GHT* structure is the *Address Search Tree* (AST), which is a structure similar to the GHT. The AST is used to actually determine the necessary (distributed) buckets when inserting and retrieving objects.

3.1 The Address Search Tree

Contrary to the GHT containing data objects in leaves, every leaf of the AST includes exactly one pointer to either a bucket (using BID) or a server (using NNID) holding the data. Specifically, NNIDs are used if the data are on a remote server. BIDs are used if the data are in a bucket on the local server. Since the clients do not maintain data buckets, their ASTs contain only the NNID pointers in leaf nodes.

A form of the AST structure is present in every network node, which naturally implies some replication. Due to the autonomous update policy, the AST structures in individual network nodes may not be identical – with respect to the complete tree view, some sub-trees may be missing. As we shall see in the next section, the GHT* provides a mechanism for updating the AST automatically during the insertion or search operations. Figure 1a illustrates the AST structure in a network of one client and two servers. The dashed arrows indicate the NNID pointers while the solid arrows represent the BID pointers.

Insert Insertion of an object starts in the node asking for insertion by traversing its AST from the root to a leaf using Condition (1). If a BID pointer is found, the inserted object is stored in this bucket. Otherwise, the found NNID pointer is applied to forward the request to the proper server where the insertion continues recursively until an AST leaf with the BID pointer is reached.

In order to avoid repeated distance computations when searching the AST on the new server, a once-determined path specification in the original AST is also forwarded. The path sent to the server is encoded as a bit-string designated BPATH, where each node is represented by one bit – "0" for the left branch, and "1" for the right branch. Due to the construction of the GHT*, it is guaranteed that the forwarded path always exists on the target server.

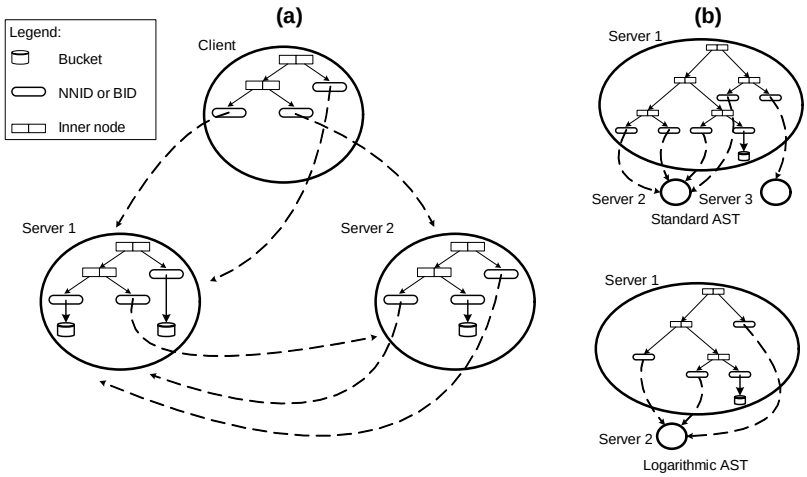


Fig. 1. AST and the GHT* network (a); example of logarithmic AST (b)

Range Search The range search also starts by traversing its local AST, but multiple paths can qualify. For all qualifying paths having a NNID pointer in their leaves, the query request with known BPATH is recursively forwarded to identified servers until a BID pointer occurs in every leaf. If multiple paths point to the same server, the request is sent only once but with multiple BPATH attachments. The range search condition is evaluated by the servers in every bucket determined by the BID pointers.

3.2 Image Adjustment

During insertion, servers split buckets without informing the other nodes of the network. Consequently, the network nodes need not have their ASTs up to date with respect to the data, but the advantage is that the network is not flooded with multiple messages at every split. The updates of the ASTs are thus postponed and actually done when respective insertion or range search operations are executed.

The inconsistency in the ASTs is recognized on a server that receives an operation request with corresponding BPATH from another client or server. In fact, if the BPATH derived from the AST of the current server is longer than the received BPATH, this indicates that the sending server (client) has an out-of-date version of the AST and must be updated. The current server easily determines a sub-tree that is missing on the sending server (client) because the root of this sub-tree is the last element of the received BPATH. Such a sub-tree is sent back to the server (client) through the *Image Adjustment Message*, IAM.

If multiple BPATHs are received by the current server, more sub-trees are sent back through one IAM (provided inconsistencies are found). Naturally, the IAM process can also involve more pairs of servers. This is a recursive procedure

which guarantees that, for an insert or a search operation, ASTs of every involved server (client) are correctly updated.

3.3 Logarithmic Replication Strategy

Using the described IAM mechanism, the GHT* structure maintains the ASTs practically equal on all servers. However, since every inner node of the AST contains two pivots, the number of replicated pivots increases linearly with the number of servers used. In order to reduce the replication, we have also implemented a much more economical strategy which achieves logarithmic replication on servers at the cost of moderately increased number of forwarded requests.

Inspired by the *lazy updates* strategy from [7], our logarithmic AST on a specific server stores only the nodes containing pointers to the local buckets (i.e. leaf nodes with BID pointers) and all their ancestors. So that the resulting AST is still a binary tree, all the sub-trees leading to leaf nodes with the NNID pointers are substituted by the leftmost leaf node of this sub-tree. The reason for choosing the leftmost leaf node is connected with our split strategy which always keeps the left node and adds the right one. Figure 1b illustrates this principle. In a way, the logarithmic AST can be seen as the minimum sub-tree of the fully updated AST. Furthermore, the image adjustment is only required when a split allocates a new bucket on a different server.

3.4 Storage Management

As we have already explained, the atomic storage unit of the GHT* is the bucket. The number of buckets and their capacity on a server are bounded by specified constant numbers, which can be different for different servers. To achieve scalability, the GHT* must be able to split buckets and allocate new storage and network resources.

Bucket Splitting The bucket splitting operation is performed in the following three steps: (1) a new bucket is allocated. If there is a capacity on the current server, the bucket is activated there. Otherwise, the bucket is allocated either on another existing server with free capacity or a new server is used; (2) a pair of pivots is chosen from the objects of the overflowing bucket. (3) objects from the overflowing bucket that are closer to the second pivot than to the first one are moved to the new bucket.

New Server Allocation In our prototype implementation, we use a pool of available servers which is known to every active server. We do not use a centralized registering service. Instead, we exploit the *broadcast messaging* to notify the active servers. When a new network node becomes available, the following actions occur: (1) the new node with its NNID sends a broadcast message saying "I am here" (this message is received by each active server in the network); (2) the receiving servers add the announced NNID to theirs local pool of available

servers. When an additional server is required, the active server picks up one item from the pool of available servers. An activation message is sent to the chosen server. With another broadcast message, the chosen server announces: "I am being used now" so that other active servers can remove its NNID from their pools of available servers. The chosen server initializes its own pool of available servers, creates a copy of the AST, and sends to the caller the "Ready to serve" reply message.

Choosing Pivots We use an incremental pivot selection algorithm from [1], which tries to find a pair of distant objects. At the beginning, the first two objects inserted into an empty bucket become the candidates for pivots. Then, we compute distances to the current candidates for every additionally inserted object. If at least one of these distances is greater than the distance between the current candidates, the new object replaces one of the candidates so that the distance between the new pair of candidates grows. When the bucket overflows, the candidates become pivots and the split is executed.

4 Performance Evaluation

In this section, we present results of performance experiments that assess different aspects of our GHT* prototype implemented in Java. The system was executed on a cluster of 100 Linux workstations connected with a 100Mbps network using the standard TCP/IP protocol.

We conducted our experiments on 45-dimensional vectors (VEC) of color image features with the L_2 (Euclidian) metric distance function and on sentences of the Czech national corpus with the *edit distance* as the metric (TXT). See [4] for more details about these data sets.

In the following, we designate the maximal number of buckets per server as n_b, and the maximal number of objects in a bucket as b_s. Due to the space limitations, we only report results concerning the range search performance and the parallel aspects of query execution.

4.1 Range Search Performance

For the range search, we have analyzed the performance with respect to different sizes of query radii. We have measured the search costs in terms of: (1) the number of servers involved in the execution of a query, (2) the number of buckets accessed, (3) the number of distance computations in the AST and in all the buckets accessed. We have not used the query execution time as the relevant criterion because we could not ensure the same environment for all participating workstations.

Experiments were executed on the GHT* structure with configuration $n_b = 10, b_s = 1,000$, which was filled with 100,000 objects either from the VEC or the TXT data-set. The average load of buckets was about 50 percent of their capacity. We have used the logarithmic replication strategy. Each point of every

graph was obtained by averaging results of 50 range queries with the same radius and a different (randomly chosen) query object.

In the first experiment, we have focused on relationships between query radius sizes and the number of buckets (servers) accessed. Figure 2 reports the results of these experiments together with the number of objects retrieved. If the radius increases, the number of accessed servers grows practically linearly, but the number of accessed buckets grows a bit faster. However, the number of retrieved objects satisfying the query grows even exponentially. This is in accordance with the behavior of centralized metric indexes such as the M-tree or the D-index on the global (not distributed) scale. The main advantages of the GHT* structure are demonstrated in Section 4.2 when the parallel execution costs are considered.

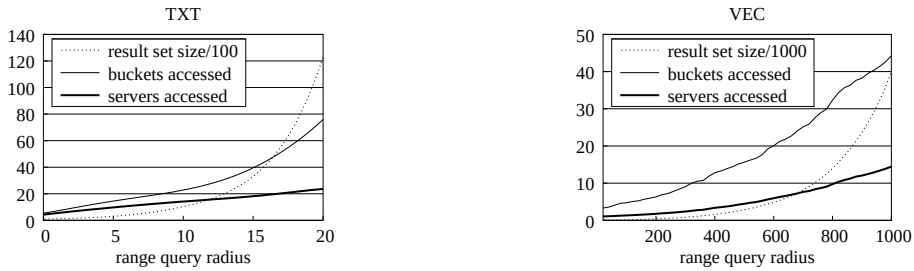


Fig. 2. Average number of buckets, servers, and retrieved objects (divided by 100 and 1,000) as a function of the radius.

Another important aspect of a distributed structure is the number of messages exchanged during search operations. Figure 3 presents the average number of messages sent by a client and the number of forwarded messages initiated on servers during a query evaluation. In fact, if we sum the number of messages sent by a client and by the servers we get the total number of servers involved in a query execution, see Figure 2 for verification. Observe that even with the logarithmic replication strategy the number of forwardings is reasonably low.

In Figure 4 we show the average number of distance computations performed by a client and the necessary servers during a query execution. We only report distance computations needed during the traversal of the AST (two distance computations must be evaluated per each inner node traversed). We do not show the number of distance computations inside the accessed buckets, because they depend on the bucket implementation strategy. In our current implementation, the buckets are organized in a dynamic list so the number of distance computations per bucket is simply given by the number of objects stored in the bucket. We are planning to use more sophisticated strategies in the future.

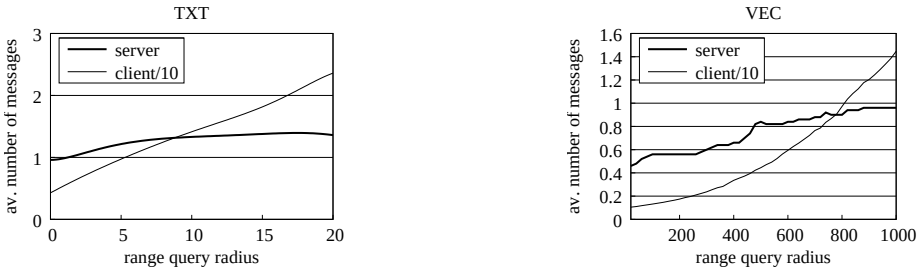


Fig. 3. Average number of messages sent by a client and server as a function of the radius.

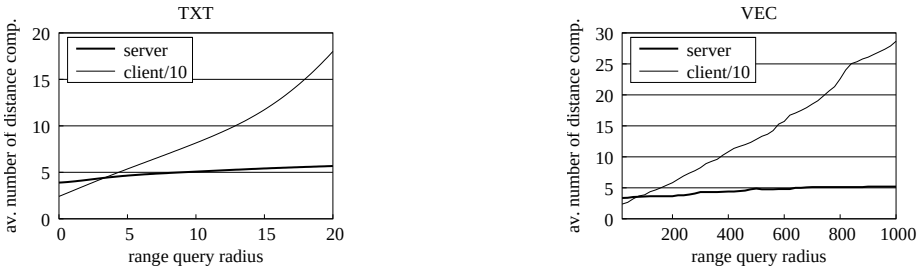


Fig. 4. Average number of distance computations in the AST for clients (divided by 10) and servers as a function of the radius.

4.2 Parallel Performance Scalability

The most important advantage of the GHT* with respect to the single-site access structures concerns its scalability through parallelism. As the size of a data-set grows, new server nodes are plugged in and their storage as well as the computing capacity is exploited. By assuming the same number of buckets of equal capacity on each server, the number of used servers grows linearly with the data-set size. This was also experimentally confirmed.

We focused on experimental studies of the intrinsic aspects of the *intraquery* and *interquery* parallelism to show how they actually contribute to the GHT* scalability. First, we consider the scalability from the perspective of a growing data-set and a fixed set of queries, i.e. the *data volume scalability*. Then, we consider a constant data-set and study how the structure scales up with the growing search radii, i.e. the *query volume scalability*.

We quantify the intraquery parallelism as the parallel response time of a range query. It is determined as the maximum of the costs incurred on servers involved in the query evaluation plus the serial search costs of the ASTs. For the evaluation purposes, we use the number of distance computations (both in the ASTs and in all the accessed buckets) as the computational costs of a query execution. In our experiments, we have neglected the communication time

because the distance computations were more time consuming than sending a message to a network node.

The interquery parallelism is more difficult to quantify. To simplify the analysis, we characterize the interquery parallelism as the ratio of the number of servers involved in a range query to the total number of servers – the lower the ratio, the higher the chances for other queries to be executed in parallel. Under the above assumptions, the intraquery parallelism is proportional to the response time of a query and the interquery parallelism represents the relative utilization of computing resources.

To evaluate the data volume scalability, we used the GHT* configuration of $n_b = 10$ $b_s = 1,000$. The graphs in Figure 5 represent the parallel search time for two different query radii as a function of the data-set size. The results are available separately for the vector (VEC) and the sentence (TXT) data-sets. By analogy, Figure 6 shows the relative utilization of servers for two types of queries and growing data-sets. The results are again averages of costs measured for 50 range queries of constant radius and different (randomly chosen) query objects.

Our experiments show that the intraquery parallelism remains very stable and the parallel search time is practically constant, i.e. independent of the data-set size. Though the number of distance computations needed for the AST traversal grows with the size of the data-set, this contribution is not visible. The reason is that the AST grow is logarithmic, while the server expansion is linear.

At the same time, the ratio characterizing the interquery parallelism in Figure 6 is even decreasing as the data-set grows in size. This means that the number of servers needed to execute the query grows much more slowly than the number of incoming active servers, thus the percentage of servers used to evaluate the query is on a down curve.

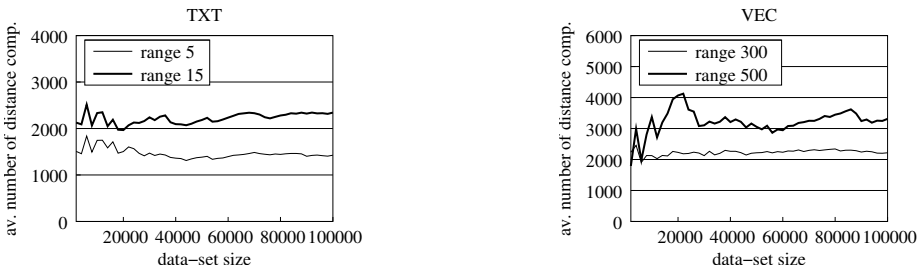


Fig. 5. Average intraquery parallelism as a function of the data-set size for two different query radii.

To evaluate the query volume scalability, we have fixed the data-set size at 100,000 objects and used queries with growing radii. Figure 7 shows the relationships between the size of a range query radius and the number of distance computations – the upper curve represents the total costs (in distance computations) to solve the query. Observe that this upper curve is closely related with

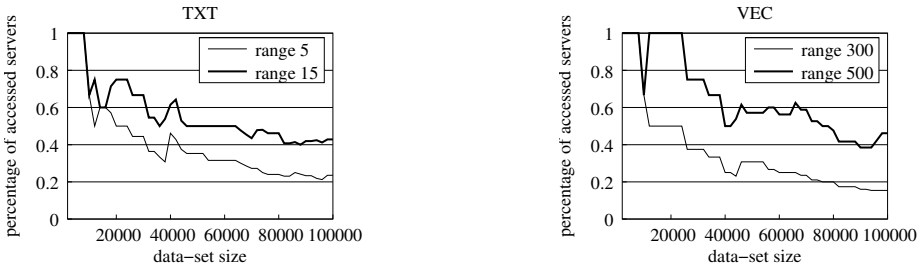


Fig. 6. Average interquery parallelism as a function of the data-set size for two different query radii.

the number of accessed buckets shown in Figure 2. The parallel cost curve represents the intraquery parallelism for the query volume scalability. Though the total computational cost of the query grows quickly as the size of the radius increases, the parallel cost remains stable after some starting phase, i.e. when retrieving very small sub-sets.

As intuitively clear, the level of the interquery parallelism for the increasing radius is actually decreasing. In fact, the larger the query radius the more servers are used from a constant set of active servers. Such property is demonstrated in Figure 2 as the linear dependence of the number of accessed servers on the radii size.

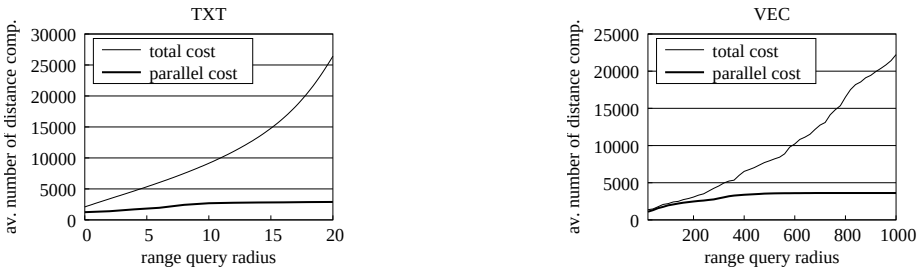


Fig. 7. Average intraquery parallelism as a function of the radius.

5 Conclusions and Future Work

To the best of our knowledge, the problem of distributed index structures for the similarity search in metric data has not been studied yet. Our structure is thus the first attempt at combining properties of scalable and distributed data structures and the principles of metric space indexing.

The GHT* structure stores and retrieves data from domains of arbitrary metric spaces and satisfies all the necessary conditions of SDDSs. It is scalable in that it distributes the structure over more and more independent servers. The parallel search time becomes practically constant for arbitrary data volume and the larger the data-set the higher the potential for the interquery parallelism. It has no hot spots – all clients and servers use as precise addressing scheme as possible and they all incrementally learn from misaddressing. Finally, updates are performed locally and a node splitting never requires sending multiple messages to many clients or servers.

The main contributions of our paper can be summarized as follows: (1) we have defined a metric scalable and distributed similarity search structure; (2) we have experimentally validated its functionality on real-life data-sets. Our future work will concentrate on strategies for updates, pre-splitting policies, and more sophisticated strategies for organizing buckets. We will also develop search algorithms for the Nearest Neighbor queries. An interesting research challenge is to consider other metric space partitioning schemes (not only the generalized hyperplane) and study their suitability for implementation in distributed environments.

References

1. B. Bustos, G. Navarro, and E. Chávez. Pivot selection techniques for proximity searching in metric spaces. In *Proc. of the XXI Conference of the Chilean Computer Science Society (SCCC'01)*, pages 33–40, 2001.
2. E. Chavez, G. Navarro, R. Baeza-Yates, and J. Marroquin. Proximity searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.
3. P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proc. of 23rd International Conference on Very Large Data Bases (VLDB)*, pages 426–435, 1997.
4. V. Dohnal, C. Gennaro, P. Savino, and P. Zezula. D-index: Distance searching index for metric data sets. *Multimedia Tools and Applications*, 21(1):9–13, 2003.
5. C. Gennaro, P. Savino, and P. Zezula. Similarity search in metric databases through hashing. In *Proc. of the 3rd International Workshop on Multimedia Information Retrieval*, pages 1–5, October 2001.
6. G. R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. *ACM Trans. Database Syst.*, 28(4):517–580, 2003.
7. T. Johnson and P. Krishna. Lazy updates for distributed search structure. In *Proc. of the ACM SIGMOD International Conference on Management of Data*, volume 22(2), pages 337–346, 1993.
8. W. Litwin, M.-A. Neimat, and D. A. Schneider. LH* - a scalable, distributed data structure. *TODS*, 21(4):480–525, 1996.
9. S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proc. of ACM SIGCOMM 2001*, pages 161–172, 2001.
10. Uhlmann. Satisfying general proximity / similarity queries with metric trees. *IPL: Information Processing Letters*, 40:175–179, 1991.

Author Index